Item 66
git_comments:

1. Default class modification
2. Set the defaults for DataTables initialisation
3. Browser
4. Bootstrap paging button renderer
5. AMD
6. ! DataTables Bootstrap 3 integration * ©2011-2015 SpryMedia Ltd - datatables.net/license
7. Because this approach is destroying and recreating the paging elements, focus is lost on the select button which is bad for accessibility. So we want to restore focus once the draw has completed
8. * * DataTables integration for Bootstrap 3. This requires Bootstrap 3 and * DataTables 1.10 or newer. * * This file sets the defaults and adds options to DataTables to style its * controls using Bootstrap. See http://datatables.net/manual/styling/bootstrap * for further information.
9. CommonJS
10. IE9 throws an 'unknown error' if document.activeElement is used inside an iframe or frame.
11. Require DataTables, which attaches to jQuery, including jQuery if needed and have a $ property so we can access the jQuery object that is used
12. * * Extend objects - very similar to jQuery.extend, but deep copy objects, and * shallow copy arrays. The reason we need to do this, is that we don't want to * deep copy array init values (such as aaSorting) since the dev wouldn't be * able to override them, but we do want to deep copy arrays. * @param {object} out Object to extend * @param {object} extender Object from which the properties will be applied to * out * @param {boolean} breakRefs If true, then arrays will be sliced to take an * independent copy with the exception of the `data` or `aaData` parameters * if they are present. This is so you can pass in a collection to * DataTables and have that used as your data source without breaking the * references * @returns {object} out Reference, just for convenience - out === the return. * @memberof DataTable#oApi * @todo This doesn't take account of arrays inside the deep copied objects.
13. * * Attach a sort listener to an element for a given column * @param {node} nNode the element to attach the sort listener to * @param {int} iColumn the column that a click on this node will sort on * @param {function} [fnCallback] callback function when sort is run * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Sort on column 1, when 'sorter' is clicked on * oTable.fnSortListener(document.getElementById('sorter'), 1); * });
14. Add search object for column specific search. Note that the `searchCols[ iCol ]` passed into extend can be undefined. This allows the user to give a default with only some of the parameters defined, and also not give a default
15. In server-side processing mode, most options are irrelevant since rows not shown don't exist and the index order is the applied order Removed is a special case - for consistency just return an empty array
16. Like the get, we need to get data from a nested object
17. Selector - function
18. Cell selector
19. Backwards compatibility pre 1.10
20. Private variable that is used to match action syntax in the data property object
21. * * An array of CSS classes that should be applied to displayed rows. This * array may be of any length, and DataTables will apply each class * sequentially, looping when required. * @type array * @default null <i>Will take the values determined by the `oClasses.stripe*` * options</i> * * @dtopt Option * @name DataTable.defaults.stripeClasses * * @example * $(document).ready(function() { * $('#example').dataTable({ * "stripeClasses": ['strip1', 'strip2', 'strip3'] * }); * })
22. This is not strict ISO8601 - Date.parse() is quite lax, although implementations differ between browsers.
23. * * Sorting data cache - this array is ostensibly the same length as the * number of columns (although each index is generated only as it is * needed), and holds the data that is used for sorting each column in the * row. We do this cache generation at the start of the sort in order that * the formatting of the sort data need be done only once for each cell * per sort. This array should not be read from or written to by anything * other than the master sorting methods. * @type array * @default null * @private
24. * * Search event, fired when the searching applied to the table (using the * built-in global search, or column filters) is altered. * @name DataTable#search.dt * @event * @param {event} e jQuery event object * @param {object} o DataTables settings object {@link DataTable.models.oSettings}
25. Check if we want to add multiple rows or not
26. 1D array
27. Not found - return an empty instance
28. * * Permanent ref to the thead element * @type node * @default null
29. Are we reading last data from DOM or the data object?
30. * * Width to expand the table to when using x-scrolling. Typically you * should not need to use this. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type string * @deprecated
31. Set No change
32. * * Deferred rendering can provide DataTables with a huge speed boost when you * are using an Ajax or JS data source for the table. This option, when set to * true, will cause DataTables to defer the creation of the table elements for * each row until they are needed for a draw - saving a significant amount of * time. * @type boolean * @default false * * @dtopt Features * @name DataTable.defaults.deferRender * * @example * $(document).ready(function() { * $('#example').dataTable({ * "ajax": "sources/arrays.txt", * "deferRender": true * }); * });
33. else
34. IE8- don't provide height and width for getBoundingClientRect
35. Create the object for storing information about this new row
36. insertBefore acts like appendChild if !arg[1]
37. Sanity check
38. Check to see if we should append an id and/or a class name to the container
39. * * Get the current page index. * * @return {integer} Current page index (zero based)
40. Selector - node
41. argument shifting
42. * * When the table is shorter in height than sScrollY, collapse the * table container down to the height of the table (when true). * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
43. * * Language compatibility - when certain options are given, and others aren't, we * need to duplicate the values over, in order to provide backwards compatibility * with older language files. * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
44. IE7 will make the width of the table when 100% include the scrollbar - which is shouldn't. When there is a scrollbar we need to take this into account.
45. * * __Deprecated__ The functionality provided by this parameter has now been * superseded by that provided through `ajax`, which should be used instead. * * This parameter allows you to override the default function which obtains * the data from the server so something more suitable for your application. * For example you could use POST data, or pull information from a Gears or * AIR database. * @type function * @member * @param {string} source HTTP source to obtain the data from (`ajax`) * @param {array} data A key/value pair object containing the data to send * to the server * @param {function} callback to be called on completion of the data get * process that will draw the data on the page. * @param {object} settings DataTables settings object * * @dtopt Callbacks * @dtopt Server-side * @name DataTable.defaults.serverData * * @deprecated 1.10. Please use `ajax` for this functionality now.
46. Remove the data property as we've resolved it already and don't want jQuery to do it again (it is restored at the end of the function)
47. * * This initialisation variable allows you to specify exactly where in the * DOM you want DataTables to inject the various controls it adds to the page * (for example you might want the pagination controls at the top of the * table). DIV elements (with or without a custom class) can also be added to * aid styling. The follow syntax is used: * <ul> * <li>The following options are allowed: * <ul> * <li>'l' - Length changing</li> * <li>'f' - Filtering input</li> * <li>'t' - The table!</li> * <li>'i' - Information</li> * <li>'p' - Pagination</li> * <li>'r' - pRocessing</li> * </ul> * </li> * <li>The following constants are allowed: * <ul> * <li>'H' - jQueryUI theme "header" classes ('fg-toolbar ui-widget-header ui-corner-tl ui-corner-tr ui-helper-clearfix')</li> * <li>'F' - jQueryUI theme "footer" classes ('fg-toolbar ui-widget-header ui-corner-bl ui-corner-br ui-helper-clearfix')</li> * </ul> * </li> * <li>The following syntax is expected: * <ul> * <li>'&lt;' and '&gt;' - div elements</li> * <li>'&lt;"class" and '&gt;' - div with a class</li> * <li>'&lt;"#id" and '&gt;' - div with an ID</li> * </ul> * </li> * <li>Examples: * <ul> * <li>'&lt;"wrapper"flipt&gt;'</li> * <li>'&lt;lf&lt;t&gt;ip&gt;'</li> * </ul> * </li> * </ul> * @type string * @default lfrtip <i>(when `jQueryUI` is false)</i>

<b>or</b> * <"H"lfr>t<"F"ip> <i>(when `jQueryUI` is true)</i> * * @dtopt Options * @name DataTable.defaults.dom * * @example * $(document).ready(function() { * $('#example').dataTable({ * "dom": '&lt;"top"i&gt;rt&lt;"bottom"flp&gt;&lt;"clear"&gt;' * }); * });

48. All properties that are available to $.fn.dataTable should also be available on $.fn.DataTable
49. Is a function - let the caller define what needs to be done
50. Scrolling from here on in
51. Class name matching on TH element
52. can't remove sorting completely
53. Width attribute
54. the original tfoot is in its own table and must be sized
55. * * Load the table state. With this function you can define from where, and how, the * state of a table is loaded. By default DataTables will load from `localStorage` * but you might wish to use a server-side database or cookies. * @type function * @member * @param {object} settings DataTables settings object * @param {object} callback Callback that can be executed when done. It * should be passed the loaded state object. * @return {object} The DataTables state object to be loaded * * @dtopt Callbacks * @name DataTable.defaults.stateLoadCallback * * @example * $(document).ready(function() { * $('#example').dataTable({ * "stateSave": true, * "stateLoadCallback": function (settings, callback) { * $.ajax({ * "url": "/state_load", * "dataType": "json", * "success": function (json) { * callback(json); * } * }); * } * }); * });
56. Information about events fired by DataTables - for documentation.
57. * * Generate the node required for filtering text * @returns {node} Filter control element * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
58. * * Callbacks for modifying the settings that have been stored for state saving * prior to using the stored values to restore the state. * @type array * @default []
59. This is very frustrating, but in IE if you just write directly to innerHTML, and elements that are overwritten are GC'ed, even if there is a reference to them elsewhere
60. Loop over each row and see if it should be included
61. Get
62. Need to translate back from the table node to the settings
63. * * This property can be used to force a DataTable to use more width than it * might otherwise do when x-scrolling is enabled. For example if you have a * table which requires to be well spaced, this parameter is useful for * "over-sizing" the table, and thus forcing scrolling. This property can by * any CSS unit, or a number (in which case it will be treated as a pixel * measurement). * @type string * @default <i>blank string - i.e. disabled</i> * * @dtopt Options * @name DataTable.defaults.scrollXInner * * @example * $(document).ready(function() { * $('#example').dataTable({ * "scrollX": "100%", * "scrollXInner": "110%" * }); * });
64. Do the sort - here we want multi-column sorting based on a given data source (column) * and sorting function (from oSort) in a certain direction. It's reasonably complex to * follow on it's own, but this is what we want (example two column sorting): * fnLocalSorting = function(a,b){ * var iTest; * iTest = oSort['string-asc']('data11', 'data12'); * if (iTest !== 0) * return iTest; * iTest = oSort['numeric-desc']('data21', 'data22'); * if (iTest !== 0) * return iTest; * return oSort['numeric-asc'](aiOrig[a], aiOrig[b]); * } * Basically we have a test for each sorting column, if the data in that column is equal, * test the next column. If all columns match, then we use a numeric sort on the row * positions in the original data array to provide a stable sort. * * Note - I know it seems excessive to have two sorting methods, but the first is around * 15% faster, so the second is only maintained for backwards compatibility with sorting * methods which do not have a pre-sort formatting function.
65. * * Get the data for the whole table, an individual row or an individual cell based on the * provided parameters. * @param {int|node} [src] A TR row node, TD/TH cell node or an integer. If given as * a TR node then the data source for the whole row will be returned. If given as a * TD/TH cell node then iCol will be automatically calculated and the data for the * cell returned. If given as an integer, then this is treated as the aoData internal * data index for the row (see fnGetPosition) and the data for that row used. * @param {int} [col] Optional column index that you want the data of. * @returns {array|object|string} If mRow is undefined, then the data for all rows is * returned. If mRow is defined, just data for that row, and is iCol is * defined, only data for the designated cell is returned. * @dtopt API * @deprecated Since v1.10 * * @example * // Row data * $(document).ready(function() { * oTable = $('#example').dataTable(); * * oTable.$('tr').click(function () { * var data = oTable.fnGetData(this); * // ... do something with the array / object of data for the row * }); * }); * * @example * // Individual cell data * $(document).ready(function() { * oTable = $('#example').dataTable(); * * oTable.$('td').click(function () { * var sData = oTable.fnGetData(this); * alert('The cell clicked on had the value of '+sData); * }); * });
66. Search the display array
67. * * This function is called on every 'draw' event, and allows you to * dynamically modify any aspect you want about the created DOM. * @type function * @param {object} settings DataTables settings object * * @dtopt Callbacks * @name DataTable.defaults.drawCallback * * @example * $(document).ready(function() { * $('#example').dataTable({ * "drawCallback": function(settings) { * alert('DataTables has redrawn the table'); * } * }); * });
68. Check if we are dealing with an array notation request
69. We know how many rows there are in the layout - so prep it
70. Note that the type is postfixed
71. For each initialisation we want to give it a clean initialisation object that can be bashed around
72. * * Set the ordering for the table. * * @param {integer} order Column index to sort upon. * @param {string} direction Direction of the sort to be applied (`asc` or `desc`) * @returns {DataTables.Api} this
73. For HTML types add a search formatter that will strip the HTML
74. Invalidate the column types as the new data needs to be revalidated
75. Unlike get, only the underscore (global) option is used for for * setting data since we don't know the type here. This is why an object * option is not documented for `mData` (which is read/write), but it is * for `mRender` which is read only.
76. * * @summary DataTables * @description Paginate, search and order HTML tables * @version 1.10.16 * @file jquery.dataTables.js * @author SpryMedia Ltd * @contact www.datatables.net * @copyright Copyright 2008-2017 SpryMedia Ltd. * * This source file is free software, available under the following license: * MIT license - http://datatables.net/license * * This source file is distributed in the hope that it will be useful, but * WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY * or FITNESS FOR A PARTICULAR PURPOSE. See the license files for details. * * For details please refer to: http://www.datatables.net
77. * * Check to see if a row is 'open' or not. * @param {node} nTr the table row to check * @returns {boolean} true if the row is currently open, false otherwise * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable; * * // 'open' an information row when a row is clicked on * $('#example tbody tr').click(function () { * if (oTable.fnIsOpen(this)) { * oTable.fnClose(this); * } else { * oTable.fnOpen(this, "Temporary row opened", "info_row"); * } * }); * * oTable = $('#example').dataTable(); * });
78. Flag to note that the table is currently being destroyed - no action should be taken
79. * * Unique identifier for each instance of the DataTables object. If there * is an ID on the table node, then it takes that value, otherwise an * incrementing internal counter is used. * @type string * @default null
80. For server-side processing tables - subtract the deleted row from the count
81. Selector options in first parameter
82. Numeric sorting types - order doesn't matter here
83. Argument shifting
84. * * Allows control over whether DataTables should use the top (true) unique * cell that is found for a single column, or the bottom (false - default). * This is useful when using complex headers. * @type boolean * @default false * * @dtopt Options * @name DataTable.defaults.orderCellsTop * * @example * $(document).ready(function() { * $('#example').dataTable({ * "orderCellsTop": true * }); * });
85. Otherwise the selector is a node, and there is one last option - the element might be a child of an element which has dt-row and dt-column data attributes
86. Indexes
87. * * Sorting that is always applied to the table (i.e. prefixed in front of * aaSorting). * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type array * @default []
88. * * The information element can be used to convey information about the current * state of the table. Although the internationalisation options presented by * DataTables are quite capable of dealing with most customisations, there may * be times where you wish to customise the string further. This callback * allows you to do exactly that. * @type function * @param {object} oSettings DataTables settings object * @param {int} start Starting position in data for the draw * @param {int} end End position in data for the draw * @param {int} max Total number of rows in the table (regardless of * filtering) * @param {int} total Total number of rows in the data set, after filtering * @param {string} pre The string that DataTables has formatted using it's * own rules * @returns {string} The string to be

displayed in the information element. * * @dtopt Callbacks * @name DataTable.defaults.infoCallback * * @example * $('#example').dataTable({ * "infoCallback": function(settings, start, end, max, total, pre) { * return start +" to "+ end; * } * });

89. * * Draw event, fired whenever the table is redrawn on the page, at the same * point as fnDrawCallback. This may be useful for binding events or * performing calculations when the table is altered at all. * @name DataTable#draw.dt * @event * @param {event} e jQuery event object * @param {object} o DataTables settings object {@link DataTable.models.oSettings}
90. * * The title of this column. * @type string * @default null <i>Derived from the 'TH' value for this column in the * original HTML table.</i> * * @name DataTable.defaults.column.title * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "title": "My column title", "targets": [0] } *] * }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* { "title": "My column title" }, * null, * null, * null, * null *] * }); * });
91. * * Buttons. For use with the Buttons extension for DataTables. This is * defined here so other extensions can define buttons regardless of load * order. It is _not_ used by DataTables core. * * @type object * @default {}
92. No need to consider sAjaxSource here since DataTables gives priority to `ajax` over `sAjaxSource`. So setting `ajax` here, renders any value of `sAjaxSource` redundant.
93. 1.11 move into sorting
94. Integer, basic index
95. Save an empty object
96. Allow an individual node to be passed in
97. `tr` element was passed in
98. Tidy the temporary table - remove name attributes so there aren't duplicated in the dom (radio elements for example)
99. * * Provide backwards compatibility for the main DT options. Note that the new * options are mapped onto the old parameters, so this is an external interface * change only. * @param {object} init Object to map
100. * * Get an array of column indexes that match a given property * @param {object} oSettings dataTables settings object * @param {string} sParam Parameter in aoColumns to look for - typically * bVisible or bSearchable * @returns {array} Array of indexes with matched properties * @memberof DataTable#oApi
101. Tell the draw function we have been filtering
102. For smart filtering we want to allow the search to work regardless of * word order. We also want double quoted text to be preserved, so word * order is important - a la google. So this is what we want to * generate: * * ^(?=.*?\bone\b)(?=.*?\btwo three\b)(?=.*?\bfour\b).*$
103. If it looks like there is an HTML entity in the string, attempt to decode it so sorting works as expected. Note that we could use a single line of jQuery to do this, but the DOM method used here is much faster http://jsperf.com/html-decode
104. And give the user a warning that we've stopped the table getting too small
105. DataTable.ext DataTable.Api DataTable.Api.register DataTable.Api.registerPlural
106. * Stripes
107. Remove any columns which are currently hidden
108. * * This function allows you to 'post process' each row after it have been * generated for each table draw, but before it is rendered on screen. This * function might be used for setting the row class name etc. * @type function * @param {node} row "TR" element for the current row * @param {array} data Raw data array for this row * @param {int} displayIndex The display index for the current table draw * @param {int} displayIndexFull The index of the data in the full list of * rows (after filtering) * * @dtopt Callbacks * @name DataTable.defaults.rowCallback * * @example * $(document).ready(function() { * $('#example').dataTable({ * "rowCallback": function(row, data, displayIndex, displayIndexFull) { * // Bold the grade for all 'A' grade browsers * if (data[4] == "A") { * $('td:eq(4)', row).html('<b>A</b>'); * } * } * }); * });
109. * * Add one or more TR elements to the table. Generally we'd expect to * use this for reading data from a DOM sourced table, but it could be * used for an TR element. Note that if a TR is given, it is used (i.e. * it is not cloned). * @param {object} settings dataTables settings object * @param {array|node|jQuery} trs The TR element(s) to add to the table * @returns {array} Array of indexes for the added rows * @memberof DataTable#oApi
110. Replace jQuery UI constants @todo depreciated
111. If the legacy.ajax parameter is null, then we automatically decide which form to use, based on sAjaxSource
112. * * DataTables can be instructed to load data to display in the table from a * Ajax source. This option defines how that Ajax call is made and where to. * * The `ajax` property has three different modes of operation, depending on * how it is defined. These are: * * * `string` - Set the URL from where the data should be loaded from. * * `object` - Define properties for `jQuery.ajax`. * * `function` - Custom data get function * * `string` * -------- * * As a string, the `ajax` property simply defines the URL from which * DataTables will load data. * * `object` * -------- * * As an object, the parameters in the object are passed to * [jQuery.ajax](http://api.jquery.com/jQuery.ajax/) allowing fine control * of the Ajax request. DataTables has a number of default parameters which * you can override using this option. Please refer to the jQuery * documentation for a full description of the options available, although * the following parameters provide additional options in DataTables or * require special consideration: * * * `data` - As with jQuery, `data` can be provided as an object, but it * can also be used as a function to manipulate the data DataTables sends * to the server. The function takes a single parameter, an object of * parameters with the values that DataTables has readied for sending. An * object may be returned which will be merged into the DataTables * defaults, or you can add the items to the object that was passed in and * not return anything from the function. This supersedes `fnServerParams` * from DataTables 1.9-. * * * `dataSrc` - By default DataTables will look for the property `data` (or * `aaData` for compatibility with DataTables 1.9-) when obtaining data * from an Ajax source or for server-side processing - this parameter * allows that property to be changed. You can use Javascript dotted * object notation to get a data source for multiple levels of nesting, or * it my be used as a function. As a function it takes a single parameter, * the JSON returned from the server, which can be manipulated as * required, with the returned value being that used by DataTables as the * data source for the table. This supersedes `sAjaxDataProp` from * DataTables 1.9-. * * * `success` - Should not be overridden it is used internally in * DataTables. To manipulate / transform the data returned by the server * use `ajax.dataSrc`, or use `ajax` as a function (see below). * * `function` * -------- * * As a function, making the Ajax call is left up to yourself allowing * complete control of the Ajax request. Indeed, if desired, a method other * than Ajax could be used to obtain the required data, such as Web storage * or an AIR database. * * The function is given four parameters and no return is required. The * parameters are: * * 1. _object_ - Data to send to the server * 2. _function_ - Callback function that must be executed when the required * data has been obtained. That data should be passed into the callback * as the only parameter * 3. _object_ - DataTables settings object for the table * * Note that this supersedes `fnServerData` from DataTables 1.9-. * * @type string|object|function * @default null * * @dtopt Option * @name DataTable.defaults.ajax * @since 1.10.0 * * @example * // Get JSON data from a file via Ajax. * // Note DataTables expects data in the form `{ data: [ ...data... ] }` by default). * $('#example').dataTable({ * "ajax": "data.json" * }); * * @example * // Get JSON data from a file via Ajax, using `dataSrc` to change * // `data` to `tableData` (i.e. `{ tableData: [ ...data... ] }`) * $('#example').dataTable({ * "ajax": { * "url": "data.json", * "dataSrc": "tableData" * } * }); * * @example * // Get JSON data from a file via Ajax, using `dataSrc` to read data * // from a plain array rather than an array in an object * $('#example').dataTable({ * "ajax": { * "url": "data.json", * "dataSrc": "" * } * }); * * @example * // Manipulate the data returned from the server - add a link to data * // (note this can, should, be done using `render` for the column - this * // is just a simple example of how the data can be manipulated). * $('#example').dataTable({ * "ajax": { * "url": "data.json", * "dataSrc": function (json) { * for (var i=0, ien=json.length ; i<ien ; i++) { * json[i][0] = '<a href="/message/'+json[i][0]+'>View message</a>'; * } * return json; * } * } * }); * * @example * // Add data to the request * $('#example').dataTable({ * "ajax": { * "url": "data.json", * "data": function (d) { * return { * "extra_search": $('#extra').val() * }; * } * } * }); * * @example * // Send request as POST * $('#example').dataTable({ * "ajax": { * "url": "data.json", * "type": "POST" * } * }); * * @example * // Get the data from localStorage (could interface with a form for * // adding, editing and removing rows). * $('#example').dataTable({ * "ajax": function (data, callback, settings) { * callback(* JSON.parse(localStorage.getItem('dataTablesData')) *); * } * });
113. * * Called at the very start of each table draw and can be used to cancel the * draw by returning false, any other return (including undefined) results in * the full draw occurring). * @type function * @param {object} settings DataTables settings object * @returns {boolean} False will cancel the draw, anything else (including no * return) will allow it to complete. * * @dtopt Callbacks * @name DataTable.defaults.preDrawCallback * * @example * $(document).ready(function() { * $('#example').dataTable({ * "preDrawCallback": function(settings) { * if ($('#test').val() == 1) { * return false; * } * } * }); * });
114. Visibility
115. ID selector. Want to always be able to select rows by id, regardless of if the tr element has been created or not, so can't rely upon jQuery here - hence a custom implementation. This does not match Sizzle's fast selector or HTML4 - in HTML5 the ID can be anything, but to select it using a CSS selector engine (like Sizzle or querySelect) it would need to need to be escaped for some characters. DataTables simplifies this for row selectors since you can select only a row. A # indicates an id any anything that follows is the id - unescaped.
116. * * Split string on periods, taking into account escaped periods * @param {string} str String to split * @return {array} Split string
117. * * Sorting that is applied to the table. Note that the inner arrays are * used in the following manner: * <ul> * <li>Index 0 - column number</li> * <li>Index 1 - current sorting direction</li> * </ul> * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type array * @todo These inner arrays should really be objects
118. CommonJS environments without a window global must pass a root. This will give an error otherwise

119. Get the remainder of the nested object to set so we can recurse
120. Table
121. * * Add a data array to the table, creating DOM node etc. This is the parallel to * _fnGatherData, but for adding rows from a Javascript source, rather than a * DOM source. * @param {object} oSettings dataTables settings object * @param {array} aData data array to be added * @param {node} [nTr] TR element to add to the table - optional. If not given, * DataTables will create a row automatically * @param {array} [anTds] Array of TD|TH elements for the row - must be given * if nTr is. * @returns {int} >=0 if successful (index of new aoData entry), -1 if failed * @memberof DataTable#oApi
122. * * See if a property is defined on one object, if so assign it to the other object * @param {object} ret target object * @param {object} src source object * @param {string} name property * @param {string} [mappedName] name to map too - optional, name used if not given * @memberof DataTable#oApi
123. Cache the setter function
124. * * Version check function. * @type function * @depreciated Since 1.10
125. * * DataTables utility methods * * This namespace provides helper methods that DataTables uses internally to * create a DataTable, but which are not exclusively used only for DataTables. * These methods can be used by extension authors to save the duplication of * code. * * @namespace
126. selector
127. * * Save the state of a table * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
128. * * Map of row ids to data indexes * @type object * @default {}
129. * * Take the column definitions and static columns arrays and calculate how * they relate to column indexes. The callback function will then apply the * definition found for a column to a suitable configuration object. * @param {object} oSettings dataTables settings object * @param {array} aoColDefs The aoColumnDefs array that is to be applied * @param {array} aoCols The aoColumns array that defines columns individually * @param {function} fn Callback function - takes two parameters, the calculated * column index and the definition for that column. * @memberof DataTable#oApi
130. Add to the display array
131. * * Redraw the tables in the current context.
132. Setting up the initialisation object
133. Property added by DT for fast lookup
134. Reduce the API instance to the first item found
135. IE9 throws an 'unknown error' if document.activeElement is used inside an iframe or frame. Try / catch the error. Not good for accessibility, but neither are frames.
136. * * Convert from camel case parameters to Hungarian, based on a Hungarian map * created by _fnHungarianMap. * @param {object} src The model object which holds all parameters that can be * mapped. * @param {object} user The object to convert from camel case to Hungarian. * @param {boolean} force When set to `true`, properties which already have a * Hungarian value in the `user` object will be overwritten. Otherwise they * won't be. * @memberof DataTable#oApi
137. * * Get the current Ajax URL. Note that this returns the URL from the first * table in the current context. * * @return {string} Current Ajax source URL
138. Add the ARIA grid role to the table
139. * * Permanent ref to the tbody element * @type node * @default null
140. * * Computed structure of the DataTables API, defined by the options passed to * `DataTable.Api.register()` when building the API. * * The structure is built in order to speed creation and extension of the Api * objects since the extensions are effectively pre-parsed. * * The array is an array of objects with the following structure, where this * base array represents the Api prototype base: * * [* { * name: 'data' -- string - Property name * val: function () {}, -- function - Api method (or undefined if just an object * methodExt: [...], -- array - Array of Api object definitions to extend the method result * propExt: [...] -- array - Array of Api object definitions to extend the property * }, * { * name: 'row' * val: {}, * methodExt: [...], * propExt: [* { * name: 'data' * val: function () {}, * methodExt: [...], * propExt: [...] * }, * ... *] * } *] * * @type {Array} * @ignore
141. Table is empty - create a row with an empty message in it
142. * * Ordering plug-ins - custom data source * * The extension options for ordering of data available here is complimentary * to the default type based ordering that DataTables typically uses. It * allows much greater control over the the data that is being used to * order a column, but is necessarily therefore more complex. * * This type of ordering is useful if you want to do ordering based on data * live from the DOM (for example the contents of an 'input' element) rather * than just the static string that DataTables knows of. * * The way these plug-ins work is that you create an array of the values you * wish to be ordering for the column in question and then return that * array. The data in the array much be in the index order of the rows in * the table (not the currently ordering order!). Which order data gathering * function is run here depends on the `dt-init columns.orderDataType` * parameter that is used for the column (if any). * * The functions defined take two parameters: * * 1. `{object}` DataTables settings object: see * {@link DataTable.models.oSettings} * 2. `{int}` Target column index * * Each function is expected to return an array: * * * `{array}` Data for the column to be ordering upon * * @type array * * @example * // Ordering using `input` node values * $.fn.dataTable.ext.order['dom-text'] = function (settings, col) * { * return this.api().column(col, {order:'index'}).nodes().map(function (td, i) { * return $('input', td).val(); * }); * }
143. * * The type allows you to specify how the data for this column will be * ordered. Four types (string, numeric, date and html (which will strip * HTML tags before ordering)) are currently available. Note that only date * formats understood by Javascript's Date() object will be accepted as type * date. For example: "Mar 26, 2008 5:03 PM". May take the values: 'string', * 'numeric', 'date' or 'html' (by default). Further types can be adding * through plug-ins. * @type string * @default null <i>Auto-detected from raw data</i> * * @name DataTable.defaults.column.type * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "type": "html", "targets": [0] } *] * }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* { "type": "html" }, * null, * null, * null, * null *] * }); * });
144. * * Text to use for the 'next' pagination button (to take the user to the * next page). * @type string * @default Next * * @dtopt Language * @name DataTable.defaults.language.paginate.next * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "paginate": { * "next": "Next page" * } * } * }); * });
145. Filter formatting functions. See model.ext.ofnSearch for information about what is required from these methods. Note that additional search methods are added for the html numbers and html formatted numbers by `_addNumericSort()` when we know what the decimal place is
146. * * Pagination string used by DataTables for the built-in pagination * control types. * @namespace * @name DataTable.defaults.language.paginate
147. * * Adjust the table column widths for new data. Note: you would probably want to * do a redraw after calling this function! * @param {object} settings dataTables settings object * @memberof DataTable#oApi
148. Clone the current header and footer elements and then place it into the inner table
149. * * Draw the header (or footer) element based on the column visibility states. The * methodology here is to use the layout array from _fnDetectHeader, modified for * the instantaneous column visibility, to construct the new layout. The grid is * traversed over cell at a time in a rows x columns grid fashion, although each * cell insert can cover multiple elements in the grid - which is tracks using the * aApplied array. Cell inserts in the grid will only occur where there isn't * already a cell in that position. * @param {object} oSettings dataTables settings object * @param array {objects} aoSource Layout array from _fnDetectHeader * @param {boolean} [bIncludeHidden=false] If true then include the hidden columns in the calc, * @memberof DataTable#oApi
150. Do a first pass on the sorting classes (allows any size changes to be taken into * account, and also will apply sorting disabled classes if disabled
151. * * Classes that DataTables assigns to the various components and features * that it adds to the HTML table. This allows classes to be configured * during initialisation in addition to through the static * {@link DataTable.ext.oStdClasses} object). * @namespace * @name DataTable.defaults.classes
152. * * State that was saved. Useful for back reference * @type object * @default null
153. * * Scrolling settings for a table. * @namespace
154. * * Get the current page length. * * @return {integer} Current page length. Note `-1` indicates that all records * are to be shown.
155. * * Information callback function. See * {@link DataTable.defaults.fnInfoCallback} * @type function * @default null
156. Does not return an API instance
157. Selector - integer
158. * * Enable or disable state saving. When enabled HTML5 `localStorage` will be * used to save table display information such as pagination information, * display length, filtering and sorting. As such when the end user reloads * the page the display display will match what thy had previously set up. * * Due to the use of `localStorage` the default state saving is not supported * in IE6 or 7. If state saving is required in those browsers, use * `stateSaveCallback` to provide a storage solution such as cookies. * @type boolean * @default false * * @dtopt Features * @name DataTable.defaults.stateSave * * @example * $(document).ready(function () { * $('#example').dataTable({ * "stateSave": true * }); * });
159. * * Indicate if all required information has been read in * @type boolean * @default false
160. * * Indicate if a redraw is being done - useful for Ajax * @type boolean * @default false
161. * * When vertical (y) scrolling is enabled, DataTables will force the height of * the table's viewport to the given height at all times (useful for layout). * However, this can look odd when filtering data down to a small data set, * and the footer is left "floating" further down. This parameter (when * enabled) will cause DataTables to collapse the table's viewport down when * the result set will fit within the given Y height. * @type boolean * @default false * * @dtopt Options *

@name DataTable.defaults.scrollCollapse * * @example * $(document).ready(function() { * $('#example').dataTable({ * "scrollY": "200", * "scrollCollapse": true * }); * });

162. Create the DOM information, or register it if already present
163. * * Class to be applied to the header element when sorting on this column - * when jQuery UI theming is used. * @type string * @default null
164. * * Callback function for the footer on each draw. * @type array * @default []
165. allow for scrolling
166. * * ARIA label that is added to the table headers when the column may be * sorted ascending by activing the column (click or return when focused). * Note that the column header is prefixed to this string. * @type string * @default : activate to sort column ascending * * @dtopt Language * @name DataTable.defaults.language.aria.sortAscending * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "aria": { * "sortAscending": " - click/return to sort ascending" * } * } * }); * });
167. * * Get the data for a given cell from the internal cache, taking into account data mapping * @param {object} settings dataTables settings object * @param {int} rowIdx aoData row id * @param {int} colIdx Column index * @param {string} type data get type ('display', 'type' 'filter' 'sort') * @returns {*} Cell data * @memberof DataTable#oApi
168. * * Show a particular column * @param {int} iCol The column whose display should be changed * @param {bool} bShow Show (true) or hide (false) the column * @param {bool} [bRedraw=true] Redraw the table or not * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Hide the second column after initialisation * oTable.fnSetColumnVis(1, false); * });
169. Run the sort by calling a full redraw
170. Remove duplicates
171. If there is a . in the source string then the data source is in a * nested object so we loop over the data for each level to get the next * level down. On each loop we test for undefined, and if found immediately * return. This allows entire objects to be missing and sDefaultContent to * be used if defined, rather than throwing an error
172. * * Check if a `<table>` node is a DataTable table already or not. * * @param {node|jquery|string} table Table node, jQuery object or jQuery * selector for the table to test. Note that if more than more than one * table is passed on, only the first will be checked * @returns {boolean} true the table given is a DataTable, or false otherwise * @static * @dtopt API-Static * * @example * if (! $.fn.DataTable.isDataTable('#example')) { * $('#example').dataTable(); * }
173. * * Fire callback functions and trigger events. Note that the loop over the * callback array store is done backwards! Further note that you do not want to * fire off triggers in time sensitive applications (for example cell creation) * as its slow. * @param {object} settings dataTables settings object * @param {string} callbackArr Name of the array storage for the callbacks in * oSettings * @param {string} eventName Name of the jQuery custom event to trigger. If * null no trigger is fired * @param {array} args Array of arguments to pass to the callback function / * trigger * @memberof DataTable#oApi
174. * * Provide backwards compatibility for column options. Note that the new options * are mapped onto the old parameters, so this is an external interface change * only. * @param {object} init Object to map
175. AMD
176. Array or flat object mapping
177. * * The classes to use for the table * @type object * @default {}
178. * * Apply custom filtering functions * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
179. Table is described by our info div
180. Correct DOM ordering for colgroup - comes before the thead
181. Might not have been created when deferred rendering
182. Pagination
183. If we get a TR element, then just add it directly - up to the dev to add the correct number of columns etc
184. * * Detail the action that will be taken when the drop down menu for the * pagination length option is changed. The '_MENU_' variable is replaced * with a default select list of 10, 25, 50 and 100, and can be replaced * with a custom select box if required. * @type string * @default Show _MENU_ entries * * @dtopt Language * @name DataTable.defaults.language.lengthMenu * * @example * // Language change only * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "lengthMenu": "Display _MENU_ records" * } * }); * }); * * @example * // Language and options change * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "lengthMenu": 'Display <select>'+ * '<option value="10">10</option>'+ * '<option value="20">20</option>'+ * '<option value="30">30</option>'+ * '<option value="40">40</option>'+ * '<option value="50">50</option>'+ * '<option value="-1">All</option>'+ * '</select> records' * } * }); * });
185. * * If restoring a table - we should restore its width * @type int * @default 0
186. can be an array of these items, comma separated list, or an array of comma separated lists
187. * * Reload tables from the Ajax data source. Note that this function will * automatically re-draw the table when the remote data has been loaded. * * @param {boolean} [reset=true] Reset (default) or hold the current paging * position. A full re-sort and re-filter is performed when this method is * called, which is why the pagination reset is the default action. * @returns {DataTables.Api} this
188. No x scrolling
189. Want argument shifting here and in _row_selector?
190. * * Generate the node required for default pagination * @param {object} oSettings dataTables settings object * @returns {node} Pagination feature node * @memberof DataTable#oApi
191. * * DataTables has a build in number formatter (`formatNumber`) which is * used to format large numbers that are used in the table information. * By default a comma is used, but this can be trivially changed to any * character you wish with this parameter. * @type string * @default , * * @dtopt Language * @name DataTable.defaults.language.thousands * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "thousands": "'" * } * }); * });
192. Default sort methods
193. * 4. Clean up
194. When scrolling we had to break the table up - restore it
195. * * Flag to indicate if the column is searchable, and thus should be included * in the filtering or not. * @type boolean
196. The min width depends upon if we have a vertical scrollbar visible or not */
197. Got the data - add it to the table
198. V8 tries _very_ hard to make a string passed into `Date.parse()` valid, so we need to use a regex to restrict date formats. Use a plug-in for anything other than ISO8601 style strings
199. Used by a lot of plug-ins, but redundant in 1.10, so this dead-end function is added to prevent errors
200. ! DataTables 1.10.16 * ©2008-2017 SpryMedia Ltd - datatables.net/license
201. * * __Deprecated__ The functionality provided by this parameter has now been * superseded by that provided through `ajax`, which should be used instead. * * You can instruct DataTables to load data from an external * source using this parameter (use aData if you want to pass data in you * already have). Simply provide a url a JSON object can be obtained from. * @type string * @default null * * @dtopt Options * @dtopt Server-side * @name DataTable.defaults.ajaxSource * * @deprecated 1.10. Please use `ajax` for this functionality now.
202. * * Attempt to load a saved table state * @param {object} oSettings dataTables settings object * @param {object} oInit DataTables init object so we can override settings * @param {function} callback Callback to execute when the state has been loaded * @memberof DataTable#oApi
203. Object to extend the base settings
204. Striping classes
205. * * Get an array of the TR nodes that are used in the table's body. Note that you will * typically want to use the '$' API method in preference to this as it is more * flexible. * @param {int} [iRow] Optional row index for the TR element you want * @returns {array|node} If iRow is undefined, returns an array of all TR elements * in the table's body, or iRow is defined, just the TR element requested. * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Get the nodes from the table * var nNodes = oTable.fnGetNodes( ); * });
206. The inner call to fetchData has already traversed through the remainder of the source requested, so we exit from the loop
207. Server-side processing init complete is done by _fnAjaxUpdateDraw
208. * * Search delay option. This will throttle full table searches that use the * DataTables provided search input element (it does not effect calls to * `dt-api search()`, providing a delay before the search is made. * @type integer * @default 0 * * @dtopt Options * @name DataTable.defaults.searchDelay * * @example * $(document).ready(function() { * $('#example').dataTable({ * "searchDelay": 200 * }); * })
209. Likewise with loading records

210. * * Defines a data source type for the ordering which can be used to read * real-time information from the table (updating the internally cached * version) prior to ordering. This allows ordering to occur on user * editable elements such as form inputs. * @type string * @default std * * @name DataTable.defaults.column.orderDataType * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "orderDataType": "dom-text", "targets": [2, 3] }, * { "type": "numeric", "targets": [3] }, * { "orderDataType": "dom-select", "targets": [4] }, * { "orderDataType": "dom-checkbox", "targets": [5] } *] * }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* null, * null, * { "orderDataType": "dom-text" }, * { "orderDataType": "dom-text", "type": "numeric" }, * { "orderDataType": "dom-select" }, * { "orderDataType": "dom-checkbox" } *] * }); * });
211. Only extend API instances and static properties of the API
212. * * Processing indicator enable flag whenever DataTables is enacting a * user request - typically an Ajax request for server-side processing. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
213. For both row and cell invalidation, the cached data for sorting and filtering is nulled out
214. * * Generate the node required for user display length changing * @param {object} settings dataTables settings object * @returns {node} Display length feature node * @memberof DataTable#oApi
215. Calculate sizes for columns
216. Bit cheeky...
217. Remove focus outline for mouse users
218. Read the ID from the DOM if present
219. * * State loaded event, fired when state has been loaded from stored data and * the settings object has been modified by the loaded data. * @name DataTable#stateLoaded.dt * @event * @param {event} e jQuery event object * @param {object} oSettings DataTables settings object * @param {object} json The saved state information
220. * * Multi-column sorting * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
221. * * Enable or disable pagination. * @type boolean * @default true * * @dtopt Features * @name DataTable.defaults.paging * * @example * $(document).ready(function () { * $('#example').dataTable({ * "paging": false * }); * });
222. * * Covert the index of an index in the data array and convert it to the visible * column index (take account of hidden columns) * @param {int} iMatch Column index to lookup * @param {object} oSettings dataTables settings object * @returns {int} i the data index * @memberof DataTable#oApi
223. Convert to 2D array if needed
224. If sorting or filtering has occurred, jump the scrolling back to the top only if we aren't holding the position
225. Last item in the input - i.e, the actual set
226. Only split on simple strings - complex expressions will be jQuery selectors
227. Sanity check that the table is of a sensible width. If not then we are going to get misalignment - try to prevent this by not allowing the table to shrink below its min width
228. * * Array of indexes for display - no filtering * @type array * @default []
229. Get the col and rowspan attributes from the DOM and sanitise them
230. Apply to the container elements
231. r1 and r2 are redundant - but it means that the parameters match for the iterator callback in columns().data()
232. * * Array of callback functions for row created function * @type array * @default []
233. Second loop once the first is done for events
234. * * Permanent ref to the tfoot element - if it exists * @type node * @default null
235. * * Update the header, footer and body tables for resizing - i.e. column * alignment. * * Welcome to the most horrible function DataTables. The process that this * function follows is basically: * 1. Re-create the table inside the scrolling div * 2. Take live measurements from the DOM * 3. Apply the measurements to align the columns * 4. Clean up * * @param {object} settings dataTables settings object * @memberof DataTable#oApi
236. if there is an ajax source load the data
237. * * This parameter is basically identical to the `sorting` parameter, but * cannot be overridden by user interaction with the table. What this means * is that you could have a column (visible or hidden) which the sorting * will always be forced on first - any sorting after that (from the user) * will then be performed as required. This can be useful for grouping rows * together. * @type array * @default null * * @dtopt Option * @name DataTable.defaults.orderFixed * * @example * $(document).ready(function() { * $('#example').dataTable({ * "orderFixed": [[0,'asc']] * }); * })
238. * * Page change event, fired when the paging of the table is altered. * @name DataTable#page.dt * @event * @param {event} e jQuery event object * @param {object} o DataTables settings object {@link DataTable.models.oSettings}
239. * 3. Apply the measurements
240. Show the display HTML options
241. Column definitions with aTargets
242. Delete from the display arrays
243. If the column is not sortable - don't to anything
244. Given that this is such a monster function, a lot of variables are use to try and keep the minimised size as small as possible
245. We don't need to do this every time DataTables is constructed, the values calculated are specific to the browser and OS configuration which we don't expect to change between initialisations
246. Backwards compatibility
247. 2D array
248. Specific renderer for this type. If available use it, otherwise use the default.
249. Add the `dt` namespace automatically if it isn't already present
250. * * Destroy event, fired when the DataTable is destroyed by calling fnDestroy * or passing the bDestroy:true parameter in the initialisation object. This * can be used to remove bound events, added DOM nodes, etc. * @name DataTable#destroy.dt * @event * @param {event} e jQuery event object * @param {object} o DataTables settings object {@link DataTable.models.oSettings}
251. * * Flag attached to the settings object so you can check in the draw * callback if filtering has been done in the draw. Deprecated in favour of * events. * @type boolean * @default false * @deprecated
252. * * Callback functions array for every time a row is inserted (i.e. on a draw). * @type array * @default []
253. * * The last jQuery XHR object that was used for server-side data gathering. * This can be used for working with the XHR information in one of the * callbacks * @type object * @default null
254. from DOM element
255. * * This function will place a new row directly after a row which is currently * on display on the page, with the HTML contents that is passed into the * function. This can be used, for example, to ask for confirmation that a * particular record should be deleted. * @param {node} nTr The table row to 'open' * @param {string|node|jQuery} mHtml The HTML to put into the row * @param {string} sClass Class to give the new TD cell * @returns {node} The row opened. Note that if the table row passed in as the * first parameter, is not found in the table, this method will silently * return. * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable; * * // 'open' an information row when a row is clicked on * $('#example tbody tr').click(function () { * if (oTable.fnIsOpen(this)) { * oTable.fnClose(this); * } else { * oTable.fnOpen(this, "Temporary row opened", "info_row"); * } * }); * * oTable = $('#example').dataTable(); * });
256. * * Callback functions for when the table has been initialised. * @type array * @default []
257. Remove old sizing and apply the calculated column widths Get the unique column headers in the newly created (cloned) header. We want to apply the calculated sizes to this header
258. * * Get the maximum strlen for each data column * @param {object} settings dataTables settings object * @param {int} colIdx column of interest * @returns {string} max string length for each column * @memberof DataTable#oApi
259. * * Defining the width of the column, this parameter may take any CSS value * (3em, 20px etc). DataTables applies 'smart' widths to columns which have not * been given a specific width through this interface ensuring that the table * remains readable. * @type string * @default null <i>Automatic</i> * * @name DataTable.defaults.column.width * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "width": "20%", "targets": [0] } *] * }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* { "width": "20%" }, * null, * null, * null, * null *] * }); * });
260. This is a little complex, but faster than always calling toString, http://jsperf.com/tostring-v-check

261. If the element we are initialising has the same ID as a table which was previously * initialised, but the table nodes don't match (from before) then we destroy the old * instance by simply deleting it. This is under the assumption that the table has been * destroyed by other methods. Anyone using non-id selectors will need to do this manually
262. Arguments passed in (list of 1D arrays)
263. Grab the data from the page - only do this when deferred loading or no Ajax * source since there is no point in reading the DOM data if we are then going * to replace it with Ajax data
264. * * Extension object for DataTables that is used to provide all extension * options. * * Note that the `DataTable.ext` object is available through * `jQuery.fn.dataTable.ext` where it may be accessed and manipulated. It is * also aliased to `jQuery.fn.dataTableExt` for historic reasons. * @namespace * @extends DataTable.models.ext
265. * * Selector extensions * * The `selector` option can be used to extend the options available for the * selector modifier options (`selector-modifier` object data type) that * each of the three built in selector types offer (row, column and cell + * their plural counterparts). For example the Select extension uses this * mechanism to provide an option to select only rows, columns and cells * that have been marked as selected by the end user (`{selected: true}`), * which can be used in conjunction with the existing built in selector * options. * * Each property is an array to which functions can be pushed. The functions * take three attributes: * * * Settings object for the host table * * Options object (`selector-modifier` object type) * * Array of selected item indexes * * The return is an array of the resulting item indexes after the custom * selector has been applied. * * @type object
266. Remove any classes added by DT_RowClass before
267. * * Sort the table by a particular column * @param {int} iCol the data index to sort on. Note that this will not match the * 'display index' if you have hidden data entries * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Sort immediately with columns 0 and 1 * oTable.fnSort([[0,'asc'], [1,'asc']]); * });
268. * * This decimal place operator is a little different from the other * language options since DataTables doesn't output floating point * numbers, so it won't ever use this for display of a number. Rather, * what this parameter does is modify the sort methods of the table so * that numbers which are in a format which has a character other than * a period (`.`) as a decimal place will be sorted numerically. * * Note that numbers with different decimal places cannot be shown in * the same table and still be sortable, the table must be consistent. * However, multiple different tables on the page can use different * decimal place characters. * @type string * @default * * @dtopt Language * @name DataTable.defaults.language.decimal * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "decimal": "," * "thousands": "." * } * }); * });
269. * Columns * See if we should load columns automatically or use defined ones
270. * * Delay the creation of TR and TD elements until they are actually * needed by a driven page draw. This can give a significant speed * increase for Ajax source and Javascript source data, but makes no * difference at all fro DOM and server-side processing tables. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
271. Sortable in both directions
272. Ensure that the table data is fully initialised
273. orderData can be given as an integer
274. Base check on table node
275. * * Generate the node required for the info display * @param {object} oSettings dataTables settings object * @returns {node} Information element * @memberof DataTable#oApi
276. Get the width of each column in the constructed table - we need to know the inner width (so it can be assigned to the other table's cells) and the outer width so we can calculate the full width of the table. This is safe since DataTables requires a unique cell for each column, but if ever a header can span multiple columns, this will need to be modified.
277. @todo Move options into their own plugins?
278. Custom sorting function - provided by the sort data type
279. Work around for Webkit bug 83867 - store the caption-side before removing from doc
280. * * When rendering large numbers in the information element for the table * (i.e. "Showing 1 to 10 of 57 entries") DataTables will render large numbers * to have a comma separator for the 'thousands' units (e.g. 1 million is * rendered as "1,000,000") to help readability for the end user. This * function will override the default method DataTables uses. * @type function * @member * @param {int} toFormat number to be formatted * @returns {string} formatted string for DataTables to show the number * * @dtopt Callbacks * @name DataTable.defaults.formatNumber * * @example * // Format a number using a single quote for the separator (note that * // this can also be done with the language.thousands option) * $(document).ready(function() { * $('#example').dataTable({ * "formatNumber": function (toFormat) { * return toFormat.toString().replace(* /\B(?=(\d{3})+(?!\d))/g, "'" *); * }; * }); * });
281. * * __Deprecated__ The functionality provided by this parameter has now been * superseded by that provided through `ajax`, which should be used instead. * * It is often useful to send extra data to the server when making an Ajax * request - for example custom filtering information, and this callback * function makes it trivial to send extra information to the server. The * passed in parameter is the data set that has been constructed by * DataTables, and you can add to this or modify it as you require. * @type function * @param {array} data Data array (array of objects which are name/value * pairs) that has been constructed by DataTables and will be sent to the * server. In the case of Ajax sourced data with server-side processing * this will be an empty array, for server-side processing there will be a * significant number of parameters! * @returns {undefined} Ensure that you modify the data array passed in, * as this is passed by reference. * * @dtopt Callbacks * @dtopt Server-side * @name DataTable.defaults.serverParams * * @deprecated 1.10. Please use `ajax` for this functionality now.
282. * * Index in the aoData array. This saves an indexOf lookup when we have the * object, but want to know the index * @type integer * @default -1 * @private
283. * * Set the current page length. * * @param {integer} Page length to set. Use `-1` to show all records. * @returns {DataTables.Api} this
284. * * Viewport width for horizontal scrolling. Horizontal scrolling is * disabled if an empty string. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type string
285. * * Store information about each column that is in use * @type array * @default []
286. Note that this isn't an exact match to the old call to _fnDraw - it takes into account the new data, but can hold position.
287. When the data source is null and a specific data type is requested (i.e. not the original data), we can use default column data
288. * * Property from a given object from which to read the table data from. This * can be an empty string (when not server-side processing), in which case * it is assumed an an array is given directly. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type string
289. * * Column sizing has changed. * @name DataTable#column-sizing.dt * @event * @param {event} e jQuery event object * @param {object} o DataTables settings object {@link DataTable.models.oSettings}
290. * * Browser feature detection for capabilities, quirks * @param {object} settings dataTables settings object * @memberof DataTable#oApi
291. Note that an int is postfixed for the sorting order
292. If not redrawing, but scrolling, we want to apply the new column sizes anyway
293. * * State save event, fired when the table has changed state a new state save * is required. This event allows modification of the state saving object * prior to actually doing the save, including addition or other state * properties (for plug-ins) or modification of a DataTables core property. * @name DataTable#stateSaveParams.dt * @event * @param {event} e jQuery event object * @param {object} oSettings DataTables settings object * @param {object} json The state information to be saved
294. Cell in the table body
295. * * DataTables features six different built-in options for the buttons to * display for pagination control: * * * `numbers` - Page number buttons only * * `simple` - 'Previous' and 'Next' buttons only * * 'simple_numbers` - 'Previous' and 'Next' buttons, plus page numbers * * `full` - 'First', 'Previous', 'Next' and 'Last' buttons * * `full_numbers` - 'First', 'Previous', 'Next' and 'Last' buttons, plus page numbers * * `first_last_numbers` - 'First' and 'Last' buttons, plus page numbers * * Further methods can be added using {@link DataTable.ext.oPagination}. * @type string * @default simple_numbers * * @dtopt Options * @name DataTable.defaults.pagingType * * @example * $(document).ready(function() { * $('#example').dataTable({ * "pagingType": "full_numbers" * }); * })
296. * * An array of data to use for the table, passed in at initialisation which * will be used in preference to any data which is already in the DOM. This is * particularly useful for constructing tables purely in Javascript, for * example with a custom Ajax call. * @type array * @default null * * @dtopt Option * @name DataTable.defaults.data * * @example * // Using a 2D array data source * $(document).ready(function () { * $('#example').dataTable({ * "data": [* ['Trident', 'Internet Explorer 4.0', 'Win 95+', 4, 'X'], * ['Trident', 'Internet Explorer 5.0', 'Win 95+', 5, 'C'], *], * "columns": [* { "title": "Engine" }, * { "title": "Browser" }, * { "title": "Platform" }, * { "title": "Version" }, * { "title": "Grade" } *] * }); * }); * * @example * // Using an array of objects as a data source (`data`) * $(document).ready(function () { * $('#example').dataTable({ * "data": [* { * "engine": "Trident", * "browser": "Internet Explorer 4.0", * "platform": "Win 95+", * "version": 4, * "grade": "X" * }, * { * "engine": "Trident", * "browser": "Internet Explorer 5.0", * "platform": "Win 95+", * "version": 5, * "grade": "C" * } *], * "columns": [* { "title": "Engine", "data": "engine" }, * { "title": "Browser", "data": "browser" }, * { "title": "Platform", "data": "platform" }, * { "title": "Version", "data": "version" }, * { "title": "Grade", "data": "grade" } *] * }); * });

297. insertBefore can act like appendChild if 2nd arg is null
298. Record set after filtering
299. * * ARIA label that is added to the table headers when the column may be * sorted descending by activing the column (click or return when focused). * Note that the column header is prefixed to this string. * @type string * @default : activate to sort column ascending * * @dtopt Language * @name DataTable.defaults.language.aria.sortDescending * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "aria": { * "sortDescending": " - click/return to sort descending" * } * } * }); * });
300. row and meta also passed, but not used
301. * * Enable or disable the display of a 'processing' indicator when the table is * being processed (e.g. a sort). This is particularly useful for tables with * large amounts of data where it can take a noticeable amount of time to sort * the entries. * @type boolean * @default false * * @dtopt Features * @name DataTable.defaults.processing * * @example * $(document).ready(function () { * $('#example').dataTable({ * "processing": true * }); * });
302. need to fall through to jQuery in case there is DOM id that matches
303. Nothing to do when the data source is null
304. * * When DataTables calculates the column widths to assign to each column, * it finds the longest string in each column and then constructs a * temporary table and reads the widths from that. The problem with this * is that "mmm" is much wider then "iiii", but the latter is a longer * string - thus the calculation can go wrong (doing it properly and putting * it into an DOM object and measuring that is horribly(!) slow). Thus as * a "work around" we provide this option. It will append its value to the * text that is found to be the longest string for the column - i.e. padding. * Generally you shouldn't need this! * @type string * @default <i>Empty string<i> * * @name DataTable.defaults.column.contentPadding * @dtopt Columns * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* null, * null, * null, * { * "contentPadding": "mmm" * } *] * }); * });
305. * * Draw index (iDraw) of the last error when parsing the returned data * @type int * @default -1
306. * * Throttle the calls to a function. Arguments and context are maintained * for the throttled function. * * @param {function} fn Function to be called * @param {integer} freq Call frequency in mS * @return {function} Wrapped function
307. In server-side processing, the draw callback will remove the processing display
308. * * You can control the default ordering direction, and even alter the * behaviour of the sort handler (i.e. only allow ascending ordering etc) * using this parameter. * @type array * @default ['asc', 'desc'] * * @name DataTable.defaults.column.orderSequence * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "orderSequence": ["asc"], "targets": [1] }, * { "orderSequence": ["desc", "asc", "asc"], "targets": [2] }, * { "orderSequence": ["desc"], "targets": [3] } *] * }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* null, * { "orderSequence": ["asc"] }, * { "orderSequence": ["desc", "asc", "asc"] }, * { "orderSequence": ["desc"] }, * null *] * }); * });
309. * Developer note: The properties of the object below are given in Hungarian * notation, that was used as the interface for DataTables prior to v1.10, however * from v1.10 onwards the primary interface is camel case. In order to avoid * breaking backwards compatibility utterly with this change, the Hungarian * version is still, internally the primary interface, but is is not documented * - hence the @name tags in each doc comment. This allows a Javascript function * to create a map from Hungarian notation to camel case (going the other direction * would require each property to be listed, which would at around 3K to the size * of DataTables, while this method is about a 0.5K hit. * * Ultimately this does pave the way for Hungarian notation to be dropped * completely, but that is a massive amount of work and will break current * installs (therefore is on-hold until v2).
310. * * Set the sorting classes on table's body, Note: it is safe to call this function * when bSort and bSortClasses are false * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
311. IE6/7 will crash if we bind a resize event handler on page load. To be removed in 1.11 which drops IE6/7 support
312. none, applied, removed applied, current, index (original - compatibility with 1.9) all, current
313. * * Array of callback functions for draw callback functions * @type array * @default []
314. * * __Deprecated__ The functionality provided by this parameter has now been * superseded by that provided through `ajax`, which should be used instead. * * By default DataTables will look for the property `data` (or `aaData` for * compatibility with DataTables 1.9-) when obtaining data from an Ajax * source or for server-side processing - this parameter allows that * property to be changed. You can use Javascript dotted object notation to * get a data source for multiple levels of nesting. * @type string * @default data * * @dtopt Options * @dtopt Server-side * @name DataTable.defaults.ajaxDataProp * * @deprecated 1.10. Please use `ajax` for this functionality now.
315. HTML5 attribute detection - build an mData object automatically if the * attributes are found
316. * * Source url for AJAX data for the table. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type string * @default null
317. * * DataTables initialisation complete event, fired when the table is fully * drawn, including Ajax data loaded, if Ajax data is required. * @name DataTable#init.dt * @event * @param {event} e jQuery event object * @param {object} oSettings DataTables settings object * @param {object} json The JSON object request from the server - only * present if client-side Ajax sourced data is used</li></ol>
318. * * Text shown inside the table records when the is no information to be * displayed after filtering. `emptyTable` is shown when there is simply no * information in the table at all (regardless of filtering). * @type string * @default No matching records found * * @dtopt Language * @name DataTable.defaults.language.zeroRecords * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "zeroRecords": "No records to display" * } * }); * });
319. jQuery or string selector
320. * * Get the number of visible columns * @param {object} oSettings dataTables settings object * @returns {int} i the number of visible columns * @memberof DataTable#oApi
321. * This is really a good bit rubbish this method of exposing the internal methods * publicly... - To be fixed in 2.0 using methods on the prototype
322. * * Nuke the table * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
323. Create the settings object for this table and set some of the default parameters
324. ARIA attributes - need to loop all columns, to update all (removing old attributes as needed)
325. Remove column sorting
326. * * Callback which allows modification of the state to be saved. Called when the table * has changed state a new state save is required. This method allows modification of * the state saving object prior to actually doing the save, including addition or * other state properties or modification. Note that for plug-in authors, you should * use the `stateSaveParams` event to save parameters for a plug-in. * @type function * @param {object} settings DataTables settings object * @param {object} data The state object to be saved * * @dtopt Callbacks * @name DataTable.defaults.stateSaveParams * * @example * // Remove a saved filter, so filtering is never saved * $(document).ready(function() { * $('#example').dataTable({ * "stateSave": true, * "stateSaveParams": function (settings, data) { * data.oSearch.sSearch = ""; * } * }); * });
327. `class` is a reserved word in Javascript, so we need to provide the ability to use a valid name for the camel case input
328. Number of columns have changed - all bets are off, no restore of settings
329. Deprecated
330. Use the draw event to trigger a callback
331. If the nested object doesn't currently exist - since we are trying to set the value - create it
332. * * Get all DataTable tables that have been initialised - optionally you can * select to get only currently visible tables. * * @param {boolean} [visible=false] Flag to indicate if you want all (default) * or visible tables only. * @returns {array} Array of `table` nodes (not DataTable instances) which are * DataTables * @static * @dtopt API-Static * * @example * $.each($.fn.dataTable.tables(true), function () { * $(table).DataTable().columns.adjust(); * });
333. * * Functions which are called prior to sending an Ajax request so extra * parameters can easily be sent to the server * @type array * @default []
334. If we need to reattach the table to the document
335. Add user defined class
336. * * Insert the required TR nodes into the table for display * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
337. Update the filtering data for each row if needed (by invalidation or first run)
338. Otherwise, if legacy has been specified then we use that to decide on the form
339. Return an Api.rows() extended instance, with the newly added row selected
340. Backwards compatibility for 1.9- which used the terminology filter rather than search
341. * * Row searching. * * This method of searching is complimentary to the default type based * searching, and a lot more comprehensive as it allows you complete control * over the searching logic. Each element in this array is a function * (parameters described below) that is called for every row in the table, * and your logic decides if it should be included in the searching data set * or not. * * Searching functions have the following input parameters: * * 1. `{object}` DataTables settings object: see * {@link DataTable.models.oSettings} * 2. `{array|object}` Data for the row to be processed (same as the * original format that was passed in as the

data source, or an array * from a DOM data source * 3. `{int}` Row index ({@link DataTable.models.oSettings.aoData}), which * can be useful to retrieve the `TR` element if you need DOM interaction. * * And the following return is expected: * * * {boolean} Include the row in the searched result set (true) or not * (false) * * Note that as with the main search ability in DataTables, technically this * is "filtering", since it is subtractive. However, for consistency in * naming we call it searching here. * * @type array * @default [] * * @example * // The following example shows custom search being applied to the * // fourth column (i.e. the data[3] index) based on two input values * // from the end-user, matching the data in a certain range. * $.fn.dataTable.ext.search.push(* function(settings, data, dataIndex) { * var min = document.getElementById('min').value * 1; * var max = document.getElementById('max').value * 1; * var version = data[3] == "-" ? 0 : data[3]*1; * * if (min == "" && max == "") { * return true; * } * else if (min == "" && version < max) { * return true; * } * else if (min < version && "" == max) { * return true; * } * else if (min < version && version < max) { * return true; * } * return false; * } *);

342. * * Apply a class to the columns which are being sorted to provide a * visual highlight or not. This can slow things down when enabled since * there is a lot of DOM interaction. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
343. Finally set the width's of the header and footer tables
344. Paging buttons
345. User specified column options
346. If there was a custom sort function, use data from there
347. * * Custom sorting data type - defines which of the available plug-ins in * afnSortData the custom sorting will use - if any is defined. * @type string * @default std
348. set
349. New search - start from the master array
350. Use a private property on the node to allow reserve mapping from the node * to the aoData array for fast look up
351. Backwards compatibility - if there is no sEmptyTable given, then use the same as * sZeroRecords - assuming that is given.
352. * * Enable or display DataTables' ability to sort multiple columns at the * same time (activated by shift-click by the user). * @type boolean * @default true * * @dtopt Options * @name DataTable.defaults.orderMulti * * @example * // Disable multiple column sorting ability * $(document).ready(function () { * $('#example').dataTable({ * "orderMulti": false * }); * });
353. * * Save the table state. This function allows you to define where and how the state * information for the table is stored By default DataTables will use `localStorage` * but you might wish to use a server-side database or cookies. * @type function * @member * @param {object} settings DataTables settings object * @param {object} data The state object to be saved * * @dtopt Callbacks * @name DataTable.defaults.stateSaveCallback * * @example * $(document).ready(function() { * $('#example').dataTable({ * "stateSave": true, * "stateSaveCallback": function (settings, data) { * // Send an Ajax request to the server with the state object * $.ajax({ * "url": "/state_save", * "data": data, * "dataType": "json", * "method": "POST" * "success": function () {} * }); * } * }); * });
354. * * Append a CSS unit (only if required) to a string * @param {string} value to css-ify * @returns {string} value with css unit * @memberof DataTable#oApi
355. Function call
356. Global filter
357. Table node
358. Add the TR elements back into the table in their original order
359. * * Text to use for the 'previous' pagination button (to take the user to * the previous page). * @type string * @default Previous * * @dtopt Language * @name DataTable.defaults.language.paginate.previous * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "paginate": { * "previous": "Previous page" * } * } * }); * });
360. * * Legacy configuration options. Enable and disable legacy options that * are available in DataTables. * * @type object
361. Use a cache array so we only need to get the type data from the formatter once (when using multiple detectors)
362. * * Data the data from the server (nuking the old) and redraw the table * @param {object} oSettings dataTables settings object * @param {object} json json data return from the server. * @param {string} json.sEcho Tracking flag for DataTables to match requests * @param {int} json.iTotalRecords Number of records in the data set, not accounting for filtering * @param {int} json.iTotalDisplayRecords Number of records in the data set, accounting for filtering * @param {array} json.aaData The data to display on this page * @param {string} [json.sColumns] Column ordering (sName, comma separated) * @memberof DataTable#oApi
363. Add the `every()` method for rows, columns and cells in a compact form
364. Get nodes in the order from the `rows` array with null values removed
365. * * Enable / disable DataTables 1.9 compatible server-side processing * requests * * @type boolean * @default null
366. * * Width of the scrollbar for the web-browser's platform. Calculated * during table initialisation. * @type int * @default 0
367. * * All of the language information can be stored in a file on the * server-side, which DataTables will look up if this parameter is passed. * It must store the URL of the language file, which is in a JSON format, * and the object has the same properties as the oLanguage object in the * initialiser object (i.e. the above parameters). Please refer to one of * the example language files to see how this works in action. * @type string * @default <i>Empty string - i.e. disabled</i> * * @dtopt Language * @name DataTable.defaults.language.url * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "url": "http://www.sprymedia.co.uk/dataTables/lang.txt" * } * }); * });
368. IE7 is throwing an error when setting these properties with jQuery's attr() and removeAttr() methods...
369. * * Element class names * * @type object * @default {}
370. * * Build a data source object from an HTML row, reading the contents of the * cells that are in the row. * * @param {object} settings DataTables settings object * @param {node|object} TR element from which to read data or existing row * object from which to re-read the data from the cells * @param {int} [colIdx] Optional column index * @param {array|object} [d] Data source object. If `colIdx` is given then this * parameter should also be given and will be used to write the data into. * Only the column in question will be written * @returns {object} Object with two parameters: `data` the data read, in * document order, and `cells` and array of nodes (they can be useful to the * caller, so rather than needing a second traversal to get them, just return * them from here). * @memberof DataTable#oApi
371. Callback functions which are array driven
372. * * Flag to say if DataTables should automatically try to calculate the * optimum table and columns widths (true) or not (false). * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
373. Update DataTables special `DT_*` attributes for the row
374. Remove the DataTables generated nodes, events and classes
375. * * Get the data from the JSON data source to use for drawing a table. Using * `_fnGetObjectDataFn` allows the data to be sourced from a property of the * source object, or from a processing function. * @param {object} oSettings dataTables settings object * @param {object} json Data source object / array from the server * @return {array} Array of data to use
376. * * Browser support parameters * @namespace
377. * * Selector for HTML tables. Apply the given selector to the give array of * DataTables settings objects. * * @param {string|integer} [selector] jQuery selector string or integer * @param {array} Array of DataTables settings objects to be filtered * @return {array} * @ignore
378. Remove column
379. jQuery functions to operate on the tables
380. * * Feature plug-ins. * * This is an array of objects which describe the feature plug-ins that are * available to DataTables. These feature plug-ins are then available for * use through the `dom` initialisation option. * * Each feature plug-in is described by an object which must have the * following properties: * * * `fnInit` - function that is used to initialise the plug-in, * * * `cFeature` - a character so the feature can be enabled by the `dom` * instillation option. This is case sensitive. * * The `fnInit` function has the following input parameters: * * 1. `{object}` DataTables settings object: see * {@link DataTable.models.oSettings} * * And the following return is expected: * * * {node|null} The element which contains your feature. Note that the * return may also be void if your plug-in does not require to inject any * DOM elements into DataTables control (`dom`) - for example this might * be useful when developing a plug-in which allows table control via * keyboard entry * * @type array * * @example * $.fn.dataTable.ext.features.push({ * "fnInit": function(oSettings) { * return new TableTools({ "oDTSettings": oSettings }); * }, * "cFeature": "T" * });
381. * * Escape a string such that it can be used in a regular expression * * @param {string} val string to escape * @returns {string} escaped string
382. Allow the data object to be passed in, or construct
383. If NaN then there isn't much formatting that we can do - just return immediately, escaping any HTML (this was supposed to be a number after all)
384. Add new column sorting
385. Depreciated The following properties are retained for backwards compatiblity only. The should not be used in new projects and will be removed in a future version
386. Remove old sorting classes
387. * * Strings that are used for WAI-ARIA labels and controls only (these are not * actually visible on the page, but will be read by screenreaders, and thus * must be internationalised as well). * @namespace * @name DataTable.defaults.language.aria

388. * * Server-side processing - number of records in the result set * (i.e. before filtering), Use fnRecordsTotal rather than * this property to get the value of the number of records, regardless of * the server-side processing setting. * @type int * @default 0 * @private
389. * * Get the number of records in the current record set, after filtering * @type function
390. ARIA role for the rows
391. * * Name for the column, allowing reference to the column by name as well as * by index (needs a lookup to work by name). * @type string
392. match by name. `names` is column index complete and in order
393. Need to take account of custom filtering functions - always filter
394. Traverse each entry in the array getting the properties requested
395. All DataTables are wrapped in a div
396. Perform a jQuery selector on the table nodes
397. * * List of options that can be used for the user selectable length menu. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type array * @default []
398. Create the cells
399. * * Format numbers for display. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type function
400. Counting from the left
401. * * Apply options for a column * @param {object} oSettings dataTables settings object * @param {int} iCol column index to consider * @param {object} oOptions object with sType, bVisible and bSearchable etc * @memberof DataTable#oApi
402. jQuery selector
403. * * Flag to indicate if the column is currently visible in the table or not * @type boolean
404. * * Developer definable function that is called whenever a cell is created (Ajax source, * etc) or processed for input (DOM source). This can be used as a compliment to mRender * allowing you to modify the DOM element (add background colour for example) when the * element is available. * @type function * @param {element} td The TD node that has been created * @param {*} cellData The Data for the cell * @param {array|object} rowData The data for the whole row * @param {int} row The row index for the aoData data store * @param {int} col The column index for aoColumns * * @name DataTable.defaults.column.createdCell * @dtopt Columns * * @example * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [{ * "targets": [3], * "createdCell": function (td, cellData, rowData, row, col) { * if (cellData == "1.7") { * $(td).css('color', 'blue') * } * } * }] * }); * });
405. Each definition can target multiple columns, as it is an array
406. Automatically invalidate
407. * * Attach a sort handler (click) to a node * @param {object} settings dataTables settings object * @param {node} attachTo node to attach the handler to * @param {int} colIdx column sorting index * @param {function} [callback] callback function * @memberof DataTable#oApi
408. * * Unique footer TH/TD element for this column (if there is one). Not used * in DataTables as such, but can be used for plug-ins to reference the * footer for each column. * @type node * @default null
409. Add to the 2D features array
410. * * Add the options to the page HTML for the table * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
411. Traverse each entry in the array setting the properties requested
412. Same again with the footer if we have one
413. * * Flag to indicate if the column is sortable or not. * @type boolean
414. * * Store for manual type assignment using the `column.type` option. This * is held in store so we can manipulate the column's `sType` property. * @type string * @default null * @private
415. * * Error reporting. * * How should DataTables report an error. Can take the value 'alert', * 'throw', 'none' or a function. * * @type string|function * @default alert
416. If there is no width attribute or style, then allow the table to collapse
417. * * Get the display end point - aiDisplay index * @type function
418. * * DataTables is a plug-in for the jQuery Javascript library. It is a highly * flexible tool, based upon the foundations of progressive enhancement, * which will add advanced interaction controls to any HTML table. For a * full list of features please refer to * [DataTables.net](href="http://datatables.net). * * Note that the `DataTable` object is not a global variable but is aliased * to `jQuery.fn.DataTable` and `jQuery.fn.dataTable` through which it may * be accessed. * * @class * @param {object} [init={}] Configuration object for DataTables. Options * are defined by {@link DataTable.defaults} * @requires jQuery 1.7+ * * @example * // Basic initialisation * $(document).ready(function { * $('#example').dataTable(); * }); * * @example * // Initialisation with configuration options - in this case, disable * // pagination and sorting. * $(document).ready(function { * $('#example').dataTable({ * "paginate": false, * "sort": false * }); * });
419. maintain local indexes
420. * * Array.prototype reduce[Right] method, used for browsers which don't support * JS 1.6. Done this way to reduce code size, since we iterate either way * @param {object} settings dataTables settings object * @memberof DataTable#oApi
421. Now do the individual column filter
422. On resort, the display master needs to be re-filtered since indexes will have changed
423. Statically defined columns array
424. v1.10 uses camelCase variables, while 1.9 uses Hungarian notation. Support both
425. * * When a user filters the information in a table, this string is appended * to the information (`info`) to give an idea of how strong the filtering * is. The variable _MAX_ is dynamically updated. * @type string * @default (filtered from _MAX_ total entries) * * @dtopt Language * @name DataTable.defaults.language.infoFiltered * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "infoFiltered": " - filtering from _MAX_ records" * } * }); * });
426. * * Quickly and simply clear a table * @param {bool} [bRedraw=true] redraw the table or not * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Immediately 'nuke' the current rows (perhaps waiting for an Ajax callback...) * oTable.fnClearTable(); * });
427. No reduce in IE8, use a loop for now
428. Remove any assigned widths from the footer (from scrolling)
429. Old parameter name of the thousands separator mapped onto the new
430. backwards compat
431. * * This string is shown in preference to `zeroRecords` when the table is * empty of data (regardless of filtering). Note that this is an optional * parameter - if it is not given, the value of `zeroRecords` will be used * instead (either the default or given value). * @type string * @default No data available in table * * @dtopt Language * @name DataTable.defaults.language.emptyTable * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "emptyTable": "No data available in table" * } * }); * });
432. * * Applied search term * @type string * @default <i>Empty string</i>
433. * * Counter for the draws that the table does. Also used as a tracker for * server-side processing * @type int * @default 0
434. * Public helper functions. These aren't used internally by DataTables, or * called by any of the options passed into DataTables, but they can be used * externally by developers working with DataTables. They are helper functions * to make working with DataTables a little bit easier.
435. fn can manipulate data or return an object object or array to merge
436. * * Store information about the table's footer * @type array * @default []
437. type, row and meta also passed, but not used
438. Need to redraw, without resorting
439. * * Build a regular expression object suitable for searching a table * @param {string} sSearch string to search for * @param {bool} bRegex treat as a regular expression or not * @param {bool} bSmart perform smart filtering or not * @param {bool} bCaseInsensitive Do case insensitive matching or not * @returns {RegExp} constructed object * @memberof DataTable#oApi
440. Store the saved state so it might be accessed at any time
441. jQuery object (also DataTables instance)
442. * * Assign a `placeholder` attribute to the search `input` element * @type string * @default * * @dtopt Language * @name DataTable.defaults.language.searchPlaceholder
443. HTML numeric
444. * * Template object for the way in which DataTables holds information about * each individual row. This is the object format used for the settings * aoData array. * @namespace

445. * * Update a table cell or row - this method will accept either a single value to * update the cell with, an array of values with one element for each column or * an object in the same format as the original data source. The function is * self-referencing in order to make the multi column updates easier. * @param {object|array|string} mData Data to update the cell/row with * @param {node|int} mRow TR element you want to update or the aoData index * @param {int} [iColumn] The column to update, give as null or undefined to * update a whole row. * @param {bool} [bRedraw=true] Redraw the table or not * @param {bool} [bAction=true] Perform pre-draw actions or not * @returns {int} 0 on success, 1 on error * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * oTable.fnUpdate('Example update', 0, 0); // Single cell * oTable.fnUpdate(['a', 'b', 'c', 'd', 'e'], $('tbody tr')[0]); // Row * });
446. * * When using Ajax sourced data and during the first draw when DataTables is * gathering the data, this message is shown in an empty row in the table to * indicate to the end user the the data is being loaded. Note that this * parameter is not used when loading data by server-side processing, just * Ajax sourced data with client-side processing. * @type string * @default Loading... * * @dtopt Language * @name DataTable.defaults.language.loadingRecords * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "loadingRecords": "Please wait - loading..." * } * }); * });
447. Selector - index
448. Calculate a layout array
449. Apply all styles in one pass
450. Rows and columns: arg1 - index arg2 - table counter arg3 - loop counter arg4 - undefined Cells: arg1 - row index arg2 - column index arg3 - table counter arg4 - loop counter
451. nb: plural might be undefined,
452. * Note that most of the paging logic is done in * DataTable.ext.pager
453. applied | removed
454. Info
455. Get scrollbar width
456. Defined else where _selector_run _selector_opts _selector_first _selector_row_indexes
457. A new instance is created if there was a selector specified
458. * * Get the widest node * @param {object} settings dataTables settings object * @param {int} colIdx column of interest * @returns {node} widest table node * @memberof DataTable#oApi
459. * * Set the current page. * * Note that if you attempt to show a page which does not exist, DataTables will * not throw an error, but rather reset the paging. * * @param {integer|string} action The paging action to take. This can be one of: * * `integer` - The page index to jump to * * `string` - An action to take: * * `first` - Jump to first page. * * `next` - Jump to the next page * * `previous` - Jump to previous page * * `last` - Jump to the last page. * @returns {DataTables.Api} this
460. * * Destroy callback functions - for plug-ins to attach themselves to the * destroy so they can clean up markup and events. * @type array * @default []
461. Insert column Need to decide if we should use appendChild or insertBefore
462. * * Width of the column * @type string * @default null
463. Numbers below, in order, are: inner.offsetWidth, inner.clientWidth, outer.offsetWidth, outer.clientWidth IE6 XP: 100 100 100 83 IE7 Vista: 100 100 100 83 IE 8+ Windows: 83 83 100 83 Evergreen Windows: 83 83 100 83 Evergreen Mac with scrollbars: 85 85 100 85 Evergreen Mac without scrollbars: 100 100 100 100
464. Browser support detection
465. * * Very similar to `columns`, `columnDefs` allows you to target a specific * column, multiple columns, or all columns, using the `targets` property of * each object in the array. This allows great flexibility when creating * tables, as the `columnDefs` arrays can be of any length, targeting the * columns you specifically want. `columnDefs` may use any of the column * options available: {@link DataTable.defaults.column}, but it _must_ * have `targets` defined in each object in the array. Values in the `targets` * array may be: * <ul> * <li>a string - class name will be matched on the TH for the column</li> * <li>0 or a positive integer - column index counting from the left</li> * <li>a negative integer - column index counting from the right</li> * <li>the string "_all" - all columns (i.e. assign a default)</li> * </ul> * @member * * @name DataTable.defaults.columnDefs
466. * * Flag to indicate if the search term should be interpreted as a * regular expression (true) or not (false) and therefore and special * regex characters escaped. * @type boolean * @default false
467. * * Order event, fired when the ordering applied to the table is altered. * @name DataTable#order.dt * @event * @param {event} e jQuery event object * @param {object} o DataTables settings object {@link DataTable.models.oSettings}
468. * * Build up the parameters in an object needed for a server-side processing * request. Note that this is basically done twice, is different ways - a modern * method which is used by default in DataTables 1.10 which uses objects and * arrays, or the 1.9- method with is name / value pairs. 1.9 method is used if * the sAjaxSource option is used in the initialisation, or the legacyAjax * option is set. * @param {object} oSettings dataTables settings object * @returns {bool} block the table drawing or not * @memberof DataTable#oApi
469. Apply all widths in final pass
470. Recursion to allow for arrays of jQuery objects
471. need to check this this is the host table, not a nested one
472. * * Use the DOM source to create up an array of header cells. The idea here is to * create a layout grid (array) of rows x columns, which contains a reference * to the cell that that point in the grid (regardless of col/rowspan), such that * any column / row could be removed and the new grid constructed * @param array {object} aLayout Array to store the calculated layout in * @param {node} nThead The header/footer element for the table * @memberof DataTable#oApi
473. `nodes` is column index complete and in order
474. All sort types have formatting functions
475. * * Template object for the column information object in DataTables. This object * is held in the settings aoColumns array and contains all the information that * DataTables needs about each individual column. * * Note that this object is related to {@link DataTable.defaults.column} * but this one is the internal data store for DataTables's cache of columns. * It should NOT be manipulated outside of DataTables. Any configuration should * be done through the initialisation options. * @namespace
476. * * Define the starting point for data display when using DataTables with * pagination. Note that this parameter is the number of records, rather than * the page number, so if you have 10 records per page and want to start on * the third page, it should be "20". * @type int * @default 0 * * @dtopt Options * @name DataTable.defaults.displayStart * * @example * $(document).ready(function() { * $('#example').dataTable({ * "displayStart": 20 * }); * })
477. * * Return an array with the full table data * @param {object} oSettings dataTables settings object * @returns array {array} aData Master data array * @memberof DataTable#oApi
478. Read the data from the DOM
479. Cache the data get and set functions for speed
480. * * Paging display length * @type int * @default 10
481. otherwise, wait for the loaded callback to be executed
482. * * Convert from camelCase notation to the internal Hungarian. We could use the * Hungarian convert function here, but this is cleaner * @param {object} obj Object to convert * @returns {object} Inverted object * @memberof DataTable#oApi
483. * * Restore the table to it's original state in the DOM by removing all of DataTables * enhancements, alterations to the DOM structure of the table and event listeners. * @param {boolean} [remove=false] Completely remove the table from the DOM * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * // This example is fairly pointless in reality, but shows how fnDestroy can be used * var oTable = $('#example').dataTable(); * oTable.fnDestroy(); * });
484. optimisation for single table case
485. Use / populate cache
486. * * Return the settings object for a particular table * @param {node} table table we are using as a dataTable * @returns {object} Settings object - or null if not found * @memberof DataTable#oApi
487. Extend with old style plug-in API methods
488. * * Enable or disable the addition of the classes `sorting\_1`, `sorting\_2` and * `sorting\_3` to the columns which are currently being sorted on. This is * presented as a feature switch as it can increase processing time (while * classes are removed and added) so for large data sets you might want to * turn this off. * @type boolean * @default true * * @dtopt Features * @name DataTable.defaults.orderClasses * * @example * $(document).ready(function () { * $('#example').dataTable({ * "orderClasses": false * }); * });
489. * * This parameter allows you to have define the global filtering state at * initialisation time. As an object the `search` parameter must be * defined, but all other parameters are optional. When `regex` is true, * the search string will be treated as a regular expression, when false * (default) it will be treated as a straight string. When `smart` * DataTables will use it's smart filtering methods (to word match at * any point in the data), when false this will not be done. * @namespace *

@extends DataTable.models.oSearch * * @dtopt Options * @name DataTable.defaults.search * * @example * $(document).ready(function() { * $('#example').dataTable({ * "search": {"search": "Initial search"} * }); * })

490. Only a single match is needed for html type since it is bottom of the pile and very similar to string
491. @todo - Is there need for an augment function? _Api.augment = function (inst, name) { // Find src object in the structure from the name var parts = name.split('.');
492. * * Get the JSON response from the last Ajax request that DataTables made to the * server. Note that this returns the JSON from the first table in the current * context. * * @return {object} JSON received from the server.
493. Initial data
494. * * Flag for if `getBoundingClientRect` is fully supported or not * @type boolean * @default false
495. * * Server-side processing - number of records in the current display set * (i.e. after filtering). Use fnRecordsDisplay rather than * this property to get the value of the number of records, regardless of * the server-side processing setting. * @type boolean * @default 0 * @private
496. Rows
497. * * TR element for the row * @type node * @default null
498. Otherwise construct a single row, worst case, table with the widest node in the data, assign any user defined widths, then insert it into the DOM and allow the browser to do all the hard work of calculating table widths don't use cloneNode - IE8 will remove events on the main table
499. * * Text which is displayed when the table is processing a user action * (usually a sort command or similar). * @type string * @default Processing... * * @dtopt Language * @name DataTable.defaults.language.processing * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "processing": "DataTables is currently busy" * } * }); * });
500. * * Flag attached to the settings object so you can check in the draw * callback if sorting has been done in the draw. Deprecated in favour of * events. * @type boolean * @default false * @deprecated
501. Apply custom sizing to the cloned header
502. _Api.extend(inst, obj); };
503. * * Private data store, containing all of the settings objects that are * created for the tables on a given page. * * Note that the `DataTable.settings` object is aliased to * `jQuery.fn.dataTableExt` through which it may be accessed and * manipulated, or `jQuery.fn.dataTable.settings`. * @member * @type array * @default [] * @private
504. * * Alter the display settings to change the page * @param {object} settings DataTables settings object * @param {string|int} action Paging action to take: "first", "previous", * "next" or "last" or page number to jump to (integer) * @param [bool] redraw Automatically draw the update or not * @returns {bool} true page has changed, false - no change * @memberof DataTable#oApi
505. Fall back - if no type was detected, always use string
506. Check and see if we have an initial draw position from state saving
507. * * Column visibility has changed. * @name DataTable#column-visibility.dt * @event * @param {event} e jQuery event object * @param {object} o DataTables settings object {@link DataTable.models.oSettings} * @param {int} column Column index * @param {bool} vis `false` if column now hidden, or `true` if visible
508. Selector - jQuery selector string, array of nodes or jQuery object/ As jQuery's .filter() allows jQuery objects to be passed in filter, it also allows arrays, so this will cope with all three options
509. Check to see if we are re-initialising a table
510. Find the widest cell for each column and put it into the table
511. Type is valid for all data points in the column - use this type
512. * * Object models container, for the various models that DataTables has * available to it. These models define the objects that are used to hold * the active state and configuration of the table. * @namespace
513. * * Abstraction for `context` parameter of the `Api` constructor to allow it to * take several different forms for ease of use. * * Each of the input parameter types will be converted to a DataTables settings * object where possible. * * @param {string|node|jQuery|object} mixed DataTable identifier. Can be one * of: * * * `string` - jQuery selector. Any DataTables' matching the given selector * with be found and used. * * `node` - `TABLE` node which has already been formed into a DataTable. * * `jQuery` - A jQuery object of `TABLE` nodes. * * `object` - DataTables settings object * * `DataTables.Api` - API instance * @return {array|null} Matching DataTables settings objects. `null` or * `undefined` is returned if no matching DataTable is found. * @ignore
514. * * Property to read the value for the cells in the column from the data * source array / object. If null, then the default content is used, if a * function is given then the return from the function is used. * @type function|int|string|null * @default null
515. * * jQuery UI class container * @type object * @deprecated Since v1.10
516. Update the input elements whenever the table is filtered
517. * * Note if draw should be blocked while getting data * @type boolean * @default true
518. Loop over the user set positioning and place the elements as needed
519. Total is tracked to remove any sub-pixel errors as the outerWidth of the table might not equal the total given here (IE!).
520. * * DataTables extensions * * This namespace acts as a collection area for plug-ins that can be used to * extend DataTables capabilities. Indeed many of the build in methods * use this method to provide their own capabilities (sorting methods for * example). * * Note that this namespace is aliased to `jQuery.fn.dataTableExt` for legacy * reasons * * @namespace
521. ? name - order?
522. * * Change the cell type created for the column - either TD cells or TH cells. This * can be useful as TH cells have semantic meaning in the table body, allowing them * to act as a header for a row (you may wish to add scope='row' to the TH elements). * @type string * @default td * * @name DataTable.defaults.column.cellType * @dtopt Columns * * @example * // Make the first column use TH cells * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [{ * "targets": [0], * "cellType": "th" * }] * }); * });
523. Cancel an existing request
524. * * Set the ordering for the table. * * @param {array} order 2D array of sorting information to be applied. * @returns {DataTables.Api} this
525. * * Enable or disable the display of this column. * @type boolean * @default true * * @name DataTable.defaults.column.visible * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "visible": false, "targets": [0] } *] }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* { "visible": false }, * null, * null, * null, * null *] }); * });
526. If the data source is a function, then we run it and use the return, executing in the scope of the data object (for instances)
527. Integer is used to pick out a table by index
528. Browser
529. All cells are going to be replaced, so empty out the row
530. Do the actual expansion in the DOM
531. * * Generate the node required for the processing node * @param {object} settings dataTables settings object * @returns {node} Processing element * @memberof DataTable#oApi
532. * * Helpers for `columns.render`. * * The options defined here can be used with the `columns.render` initialisation * option to provide a display renderer. The following functions are defined: * * * `number` - Will format numeric data (defined by `columns.data`) for * display, retaining the original unformatted data for sorting and filtering. * It takes 5 parameters: * * `string` - Thousands grouping separator * * `string` - Decimal point indicator * * `integer` - Number of decimal points to show * * `string` (optional) - Prefix. * * `string` (optional) - Postfix (/suffix). * * `text` - Escape HTML to help prevent XSS attacks. It has no optional * parameters. * * @example * // Column definition using the number renderer * { * data: "salary", * render: $.fn.dataTable.render.number('\'', '.', 0, '$') * } * * @namespace
533. If there is col / rowspan, copy the information into the layout grid
534. * * Draw the table for the first time, adding all required features * @param {object} settings dataTables settings object * @memberof DataTable#oApi
535. Built in type detection. See model.ext.aTypes for information about what is required from this methods.
536. Copy the data index array
537. Can't just check for isArray here, as an API or jQuery instance might be given with their array like look
538. * * Callbacks for modifying the settings to be stored for state saving, prior to * saving state. * @type array * @default []
539. Existing row object passed in
540. * * Type based search formatting. * * The type based searching functions can be used to pre-format the * data to be search on. For example, it can be used to strip HTML * tags or to de-format telephone numbers for numeric only searching. * * Note that is a search is not defined for a column of a given type, * no search formatting will be performed. * * Pre-processing of searching data plug-ins - When you assign the sType * for a column (or have it automatically detected for you by DataTables * or a type detection plug-in), you will typically be using this for * custom sorting, but it can also be used to provide custom searching * by allowing

you to pre-processing the data and returning the data in * the format that should be searched upon. This is done by adding * functions this object with a parameter name which matches the sType * for that target column. This is the corollary of <i>afnSortData</i> * for searching data. * * The functions defined take a single parameter: * * 1. `{*}` Data from the column cell to be prepared for searching * * Each function is expected to return: * * * `{string|null}` Formatted string that will be used for the searching. * * @type object * @default {} * * @example * $.fn.dataTable.ext.type.search['title-numeric'] = function (d) { * return d.replace(/\n/g," ").replace(/<.*?>/g, ""); * }

541. Map camel case parameters to their Hungarian counterparts
542. * * When enabled DataTables will not make a request to the server for the first * page draw - rather it will use the data already on the page (no sorting etc * will be applied to it), thus saving on an XHR at load time. `deferLoading` * is used to indicate that deferred loading is required, but it is also used * to tell DataTables how many records there are in the full table (allowing * the information element and pagination to be displayed correctly). In the case * where a filtering is applied to the table on initial load, this can be * indicated by giving the parameter as an array, where the first element is * the number of records available after filtering and the second element is the * number of records without filtering (allowing the table information element * to be shown correctly). * @type int | array * @default null * * @dtopt Options * @name DataTable.defaults.deferLoading * * @example * // 57 records available in the table, no filtering applied * $(document).ready(function() { * $('#example').dataTable({ * "serverSide": true, * "ajax": "scripts/server_processing.php", * "deferLoading": 57 * }); * }); * * @example * // 57 records after filtering, 100 without filtering (an initial filter applied) * $(document).ready(function() { * $('#example').dataTable({ * "serverSide": true, * "ajax": "scripts/server_processing.php", * "deferLoading": [57, 100], * "search": { * "search": "my_filter" * } * }); * });
543. Array results are 'enhanced'
544. Fire length change event
545. Server-side processing draw intercept
546. * * Array of TD elements for each row. This is null until the row has been * created. * @type array nodes * @default []
547. IE9 throws an 'unknown error' if document.activeElement is used inside an iframe or frame...
548. Use getBounding... where possible (not IE8-) because it can give sub-pixel accuracy, which we then want to round up!
549. Remove the row's ID reference if there is one
550. * * Paging start point - aiDisplay index * @type int * @default 0
551. * * Title of the column - what is seen in the TH element (nTh). * @type string
552. Build an object of get functions, and wrap them in a single call
553. Yes, modify the sort
554. * * Index for what 'this' index API functions should use * @type int * @deprecated Since v1.10
555. When data was added after the initialisation (data or Ajax) we need to calculate the column sizing
556. * * Type detection functions. * * The functions defined in this object are used to automatically detect * a column's type, making initialisation of DataTables super easy, even * when complex data is in the table. * * The functions defined take two parameters: * * 1. `{*}` Data from the column cell to be analysed * 2. `{settings}` DataTables settings object. This can be used to * perform context specific type detection - for example detection * based on language settings such as using a comma for a decimal * place. Generally speaking the options from the settings will not * be required * * Each function is expected to return: * * * `{string|null}` Data type detected, or null if unknown (and thus * pass it on to the other type detection functions. * * @type array * * @example * // Currency type detection plug-in: * $.fn.dataTable.ext.type.detect.push(* function (data, settings) { * // Check the numeric part * if (! $.isNumeric(data.substring(1))) { * return null; * } * * // Check prefixed by currency * if (data.charAt(0) == '$' || data.charAt(0) == '&pound;') { * return 'currency'; * } * return null; * } *);
557. * * Software version * @type string * @deprecated Since v1.10
558. Attach a sort listener to update on sort
559. * * Initialisation options that can be given to DataTables at initialisation * time. * @namespace
560. Process each column
561. Draw is complete, sorting and filtering must be as well
562. Track if we can use the fast sort algorithm
563. * * Function to run on user sort request * @param {object} settings dataTables settings object * @param {node} attachTo node to attach the handler to * @param {int} colIdx column sorting index * @param {boolean} [append=false] Append the requested sort to the existing * sort if true (i.e. multi-column sort) * @param {function} [callback] callback function * @memberof DataTable#oApi
564. * * Paging information for the first table in the current context. * * If you require paging information for another table, use the `table()` method * with a suitable selector. * * @return {object} Object with the following properties set: * * `page` - Current page index (zero based - i.e. the first page is `0`) * * `pages` - Total number of pages * * `start` - Display index for the first record shown on the current page * * `end` - Display index for the last record shown on the current page * * `length` - Display length (number of records). Note that generally `start * + length = end`, but this is not always true, for example if there are * only 2 records to show on the final page, with a length of 10. * * `recordsTotal` - Full data set length * * `recordsDisplay` - Data set length once the current filtering criterion * are applied.
565. * * Information about open rows. Each object in the array has the parameters * 'nTr' and 'nParent' * @type array * @default []
566. else return undefined;
567. Reject old data
568. else, have an action to take on all tables
569. Convert the camel-case defaults to Hungarian
570. * * Change the order of the table * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi * @todo This really needs split up!
571. Special parameters can be given by the data source to be used on the row
572. * * Change the pagination - provides the internal logic for pagination in a simple API * function. With this function you can have a DataTables table go to the next, * previous, first or last pages. * @param {string|int} mAction Paging action to take: "first", "previous", "next" or "last" * or page number to jump to (integer), note that page 0 is the first page. * @param {bool} [bRedraw=true] Redraw the table or not * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * oTable.fnPageChange('next'); * });
573. Number of number buttons (including ellipsis) to show. _Must be odd!_
574. Get the remainder of the nested object to get
575. * * Throttle the calls to a function. Arguments and context are maintained for * the throttled function * @param {function} fn Function to be called * @param {int} [freq=200] call frequency in mS * @returns {function} wrapped function * @memberof DataTable#oApi
576. html
577. Convert to object based for 1.10+ if using the old array scheme which can come from server-side processing or serverParams
578. When scrolling (X or Y) we want to set the width of the table as appropriate. However, when not scrolling leave the table width as it is. This results in slightly different, but I think correct behaviour
579. * * Indicate if when using server-side processing the loading of data * should be deferred until the second draw. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean * @default false
580. * * Draw the table for the first time, adding all required features * @param {object} oSettings dataTables settings object * @param {object} [json] JSON from the server that completed the table, if using Ajax source * with client-side processing (optional) * @memberof DataTable#oApi
581. Boolean initialisation of x-scrolling
582. Otherwise a node which might have a `dt-column` data attribute, or be a child or such an element
583. * * Function to set data for a cell in the column. You should <b>never</b> * set the data directly to _aData internally in DataTables - always use * this method. It allows mData to function as required. This function * is automatically assigned by the column initialisation method * @type function * @param {array|object} oData The data array/object for the array * (i.e. aoData[]._aData) * @param {*} sValue Value to set * @default null
584. * * Text to use when using the 'full_numbers' type of pagination for the * button to take the user to the last page. * @type string * @default Last * * @dtopt Language * @name DataTable.defaults.language.paginate.last * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "paginate": { * "last": "Last page" * } * } * }); * });
585. Selector - all
586. Remove once webkit bug 131819 and Chromium bug 365619 have been resolved and deployed
587. * * By default DataTables allows keyboard navigation of the table (sorting, paging, * and filtering) by adding a `tabindex` attribute to the required elements. This * allows you to tab through the controls and press the enter key to activate them. * The tabindex is default 0, meaning that the tab follows the flow of the document. * You can overrule this using this parameter if you wish. Use a value of -1 to * disable built-in keyboard navigation. * @type int * @default 0 * * @dtopt Options * @name DataTable.defaults.tabIndex * * @example * $(document).ready(function() { * $('#example').dataTable({ * "tabIndex": 1 * }); * });
588. Search in DataTables 1.10 is string based. In 1.11 this should be altered to also allow strict type checking.

589. Figure out if there are scrollbar present - if so then we need a the header and footer to provide a bit more space to allow "overflow" scrolling (i.e. past the scrollbar)
590. Prevent form submission
591. Read data from a cell and store into the data object
592. * * Data object from the original data source for the row. This is either * an array if using the traditional form of DataTables, or an object if * using mData options. The exact type will depend on the passed in * data from the data source, or will be an array if using DOM a data * source. * @type array|object * @default []
593. Array notation
594. Store the data submitted for the API
595. If the function returned something, use that alone
596. legacy x scroll inner has been given - use it
597. Object with rows, columns and opts
598. No sort on this column yet
599. * * Filter the table on a per-column basis * @param {object} oSettings dataTables settings object * @param {string} sInput string to filter on * @param {int} iColumn column to filter * @param {bool} bRegex treat search string as a regular expression or not * @param {bool} bSmart use smart filtering or not * @param {bool} bCaseInsensitive Do case insenstive matching or not * @memberof DataTable#oApi
600. In ARIA only the first sorting column can be marked as sorting - no multi-sort option
601. * * Update the table using an Ajax call * @param {object} settings dataTables settings object * @returns {boolean} Block the table drawing or not * @memberof DataTable#oApi
602. * * Perform a jQuery selector action on the table's TR elements (from the tbody) and * return the resulting jQuery object. * @param {string|node|jQuery} sSelector jQuery selector or node collection to act on * @param {object} [oOpts] Optional parameters for modifying the rows to be included * @param {string} [oOpts.filter=none] Select TR elements that meet the current filter * criterion ("applied") or all TR elements (i.e. no filter). * @param {string} [oOpts.order=current] Order of the TR elements in the processed array. * Can be either 'current', whereby the current sorting of the table is used, or * 'original' whereby the original order the data was read into the table is used. * @param {string} [oOpts.page=all] Limit the selection to the currently displayed page * ("current") or not ("all"). If 'current' is given, then order is assumed to be * 'current' and filter is 'applied', regardless of what they might be given as. * @returns {object} jQuery object, filtered by the given selector. * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Highlight every second row * oTable.$('tr:odd').css('backgroundColor', 'blue'); * }); * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Filter to rows with 'Webkit' in them, add a background colour and then * // remove the filter, thus highlighting the 'Webkit' rows only. * oTable.fnFilter('Webkit'); * oTable.$('tr', {"search": "applied"}).css('backgroundColor', 'blue'); * oTable.fnFilter(''); * });
603. Internal data grab
604. Provide a pre-callback function which can be used to cancel the draw is false is returned
605. * * Data location where to store a row's id * @type string * @default null
606. If appending the sort then we are multi-column sorting
607. * * Create a new TR element (and it's TD children) for a row * @param {object} oSettings dataTables settings object * @param {int} iRow Row to consider * @param {node} [nTrIn] TR element to add to the table - optional. If not given, * DataTables will create a row automatically * @param {array} [anTds] Array of TD|TH elements for the row - must be given * if nTr is. * @memberof DataTable#oApi
608. * * Function to get data from a cell in a column. You should <b>never</b> * access data directly through _aData internally in DataTables - always use * the method attached to this property. It allows mData to function as * required. This function is automatically assigned by the column * initialisation method * @type function * @param {array|object} oData The data array/object for the array * (i.e. aoData[]._aData) * @param {string} sSpecific The specific data type you want to get - * 'display', 'type' 'filter' 'sort' * @returns {*} The data for the cell from the given row's data * @default null
609. Prep the applied array - it needs an element for each row
610. Cache the footer cells. Note that we only take the cells from the first row in the footer. If there is more than one row the user wants to interact with, they need to use the table().foot() method. Note also this allows cells to be used for multiple columns using colspan
611. remove
612. * * Callback which allows modification of the saved state prior to loading that state. * This callback is called when the table is loading state from the stored data, but * prior to the settings object being modified by the saved state. Note that for * plug-in authors, you should use the `stateLoadParams` event to load parameters for * a plug-in. * @type function * @param {object} settings DataTables settings object * @param {object} data The state object that is to be loaded * * @dtopt Callbacks * @name DataTable.defaults.stateLoadParams * * @example * // Remove a saved filter, so filtering is never loaded * $(document).ready(function() { * $('#example').dataTable({ * "stateSave": true, * "stateLoadParams": function (settings, data) { * data.oSearch.sSearch = ""; * } * }); * }); * * @example * // Disallow state loading by returning false * $(document).ready(function() { * $('#example').dataTable({ * "stateSave": true, * "stateLoadParams": function (settings, data) { * return false; * } * }); * });
613. * * Add attributes to a row based on the special `DT_*` parameters in a data * source object. * @param {object} settings DataTables settings object * @param {object} DataTables row object for the row to be modified * @memberof DataTable#oApi
614. Cache calculation for unique columns
615. * * This parameter is only used in DataTables' server-side processing. It can * be exceptionally useful to know what columns are being displayed on the * client side, and to map these to database fields. When defined, the names * also allow DataTables to reorder information from the server if it comes * back in an unexpected order (i.e. if you switch your columns around on the * client-side, your server-side code does not also need updating). * @type string * @default <i>Empty string</i> * * @name DataTable.defaults.column.name * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "name": "engine", "targets": [0] }, * { "name": "browser", "targets": [1] }, * { "name": "platform", "targets": [2] }, * { "name": "version", "targets": [3] }, * { "name": "grade", "targets": [4] } *] * }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* { "name": "engine" }, * { "name": "browser" }, * { "name": "platform" }, * { "name": "version" }, * { "name": "grade" } *] * }); * });
616. Update node value whenever anything changes the table's length
617. * * This parameter allows you to readily specify the entries in the length drop * down menu that DataTables shows when pagination is enabled. It can be * either a 1D array of options which will be used for both the displayed * option and the value, or a 2D array which will use the array in the first * position as the value, and the array in the second position as the * displayed options (useful for language strings such as 'All'). * * Note that the `pageLength` property will be automatically set to the * first value given in this array, unless `pageLength` is also provided. * @type array * @default [10, 25, 50, 100] * * @dtopt Option * @name DataTable.defaults.lengthMenu * * @example * $(document).ready(function() { * $('#example').dataTable({ * "lengthMenu": [[10, 25, 50, -1], [10, 25, 50, "All"]] * }); * });
618. * * Mark cached data as invalid such that a re-read of the data will occur when * the cached data is next requested. Also update from the data source object. * * @param {object} settings DataTables settings object * @param {int} rowIdx Row index to invalidate * @param {string} [src] Source to invalidate from: undefined, 'auto', 'dom' * or 'data' * @param {int} [colIdx] Column index to invalidate. If undefined the whole * row will be invalidated * @memberof DataTable#oApi * * @todo For the modularisation of v1.11 this will need to become a callback, so * the sort and filter methods can subscribe to it. That will required * initialisation options for sorting, which is why it is not already baked in
619. For scrollX we need to force the column width otherwise the browser will collapse it. If this width is smaller than the width the column requires, then it will have no effect
620. * * The state duration (for `stateSave`) in seconds. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type int * @default 0
621. Copy the camelCase options over to the hungarian
622. Use the default
623. Update the cached indexes
624. * * Last applied sort * @type array * @default []
625. Allow a jQuery object to be passed in - only a single row is added from it though - the first element in the set
626. Filter
627. IE6/7 will oversize a width 100% element inside a scrolling element, to include the width of the scrollbar, while other browsers ensure the inner element is contained without forcing scrolling
628. http://en.wikipedia.org/wiki/Foreign_exchange_market - \u20BD - Russian ruble. - \u20a9 - South Korean Won - \u20BA - Turkish Lira - \u20B9 - Indian Rupee - R - Brazil (R$) and South Africa - fr - Swiss Franc - kr - Swedish krona, Norwegian krone and Danish krone - \u2009 is thin space and \u202F is narrow no-break space, both used in many standards as thousands separators.

629. Plain numbers - first since V8 detects some plain numbers as dates e.g. Date.parse('55') (but not all, e.g. Date.parse('22')...).
630. Call all required callback functions for the end of a draw
631. jQuery access
632. * * Provide a common method for plug-ins to check the version of DataTables being used, in order * to ensure compatibility. * @param {string} sVersion Version string to check for, in the format "X.Y.Z". Note that the * formats "X" and "X.Y" are also acceptable. * @returns {boolean} true if this version of DataTables is greater or equal to the required * version, or false if this version of DataTales is not suitable * @method * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * alert(oTable.fnVersionCheck('1.9.0')); * });
633. Dates
634. * * Ajax (XHR) event, fired whenever an Ajax request is completed from a * request to made to the server for new data. This event is called before * DataTables processed the returned data, so it can also be used to pre- * process the data returned from the server, if needed. * * Note that this trigger is called in `fnServerData`, if you override * `fnServerData` and which to use this event, you need to trigger it in you * success function. * @name DataTable#xhr.dt * @event * @param {event} e jQuery event object * @param {object} o DataTables settings object {@link DataTable.models.oSettings} * @param {object} json JSON returned from the server * * @example * // Use a custom property returned from the server in another DOM element * $('#table').dataTable().on('xhr.dt', function (e, settings, json) { * $('#status').html(json.status); * }); * * @example * // Pre-process the data returned from the server * $('#table').dataTable().on('xhr.dt', function (e, settings, json) { * for (var i=0, ien=json.aaData.length ; i<ien ; i++) { * json.aaData[i].sum = json.aaData[i].one + json.aaData[i].two; * } * // Note no return - manipulate the data directly in the JSON object. * });
635. Trigger xhr
636. Add column to aoColumns array
637. If null, then this type can't apply to this column, so rather than testing all cells, break out. There is an exception for the last type which is `html`. We need to scan all rows since it is possible to mix string and HTML types
638. Create a value - key array of the current row positions such that we can use their current position during the sort, if values match, in order to perform stable sorting
639. Page length
640. * * Determine if the vertical scrollbar is on the right or left of the * scrolling container - needed for rtl language layout, although not * all browsers move the scrollbar (Safari). * @type boolean * @default false
641. IE7< puts a vertical scrollbar in place (when it shouldn't be) due to subtracting * the scrollbar height from the visible display, rather than adding it on. We need to * set the height in order to sort this. Don't want to do it in any other browsers.
642. * * Column options that can be given to DataTables at initialisation time. * @namespace
643. Check if there is data passing into the constructor
644. global oInit,_that,emptyInit
645. * * Cache the wrapper node (contains all DataTables controlled elements) * @type node * @default null
646. Backwards compatibility for the defaults
647. meta is also passed in, but not used
648. Dates (only those recognised by the browser's Date.parse)
649. Need to convert back to 1.9 trad format
650. Move along the position array
651. * * Server-side processing enabled flag - when enabled DataTables will * get all data from the server for every draw - there is no filtering, * sorting or paging done on the client-side. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
652. Build and draw the header / footer for the table
653. If the scrollbar visibility has changed from the last draw, we need to adjust the column sizes as the table width will have changed to account for the scrollbar
654. Okay to show that something is going on now
655. * * Convert a CSS unit width to pixels (e.g. 2em) * @param {string} width width to be converted * @param {node} parent parent to get the with for (required for relative widths) - optional * @returns {int} width in pixels * @memberof DataTable#oApi
656. applied search removed search
657. Set
658. Allow custom and plug-in manipulation functions to alter the saved data set and cancelling of loading by returning false
659. * * Text to use when using the 'full_numbers' type of pagination for the * button to take the user to the first page. * @type string * @default First * * @dtopt Language * @name DataTable.defaults.language.paginate.first * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "paginate": { * "first": "First page" * } * } * }); * });
660. Add the numeric 'deformatting' functions for sorting and search. This is done in a function to provide an easy ability for the language options to add additional methods if a non-period decimal place is used.
661. Clean up the table body
662. Columns
663. If we are a scrolling table, and no footer has been given, then we need to create a tfoot element for the caption element to be appended to
664. New API instance returned, want the value from the first item in the returned array for the singular result.
665. * * Get current ordering (sorting) that has been applied to the table. * * @returns {array} 2D array containing the sorting information for the first * table in the current context. Each element in the parent array represents * a column being sorted upon (i.e. multi-sorting with two columns would have * 2 inner arrays). The inner arrays may have 2 or 3 elements. The first is * the column index that the sorting condition applies to, the second is the * direction of the sort (`desc` or `asc`) and, optionally, the third is the * index of the sorting order from the `column.sorting` initialisation array.
666. IE6/7 are a law unto themselves...
667. * * Store information about the table's header * @type array * @default []
668. * * All strings that DataTables uses in the user interface that it creates * are defined in this object, allowing you to modified them individually or * completely replace them all as required. * @namespace * @name DataTable.defaults.language
669. Built our DOM structure - replace the holding div with what we want
670. If the children were already shown, that state should be retained
671. jQuery's factory checks for a global window
672. Check if any of the rows were invalidated
673. Update the colspan for the details rows (note, only if it already has a colspan)
674. jQuery selector on the TH elements for the columns
675. * * Retrieve the DataTables object for the given selector. Note that if the * table has already been initialised, this parameter will cause DataTables * to simply return the object that has already been set up - it will not take * account of any changes you might have made to the initialisation object * passed to DataTables (setting this parameter to true is an acknowledgement * that you understand this). `destroy` can be used to reinitialise a table if * you need. * @type boolean * @default false * * @dtopt Options * @name DataTable.defaults.retrieve * * @example * $(document).ready(function() { * initTable(); * tableActions(); * }); * * function initTable () * { * return $('#example').dataTable({ * "scrollY": "200px", * "paginate": false, * "retrieve": true * }); * } * * function tableActions () * { * var table = initTable(); * // perform API operations with oTable * }
676. * * Present a user control allowing the end user to change the page size * when pagination is enabled. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
677. mental IE8 fix :-(
678. * * The TABLE node for the main table * @type node * @default null
679. * * Search delay (in mS) * @type integer * @default null
680. Backwards compatibility. Alias to pre 1.10 Hungarian notation counter parts
681. * * Identical to fnHeaderCallback() but for the table footer this function * allows you to modify the table footer on every 'draw' event. * @type function * @param {node} foot "TR" element for the footer * @param {array} data Full table data (as derived from the original HTML) * @param {int} start Index for the current display starting point in the * display array * @param {int} end Index for the current display ending point in the * display array * @param {array int} display Index array to translate the visual position * to the full data array * * @dtopt Callbacks * @name DataTable.defaults.footerCallback * * @example * $(document).ready(function() { * $('#example').dataTable({ * "footerCallback": function(tfoot, data, start, end, display) { * tfoot.getElementsByTagName('th')[0].innerHTML = "Starting index is "+start; * } * }); * })
682. HTML (this is strict checking - there must be html)
683. * * Callback functions for just before the table is redrawn. A return of * false will be used to cancel the draw. * @type array * @default []

684. * * Processing event, fired when DataTables is doing some kind of processing * (be it, order, searcg or anything else). It can be used to indicate to * the end user that there is something happening, or that something has * finished. * @name DataTable#processing.dt * @event * @param {event} e jQuery event object * @param {object} oSettings DataTables settings object * @param {boolean} bShow Flag for if DataTables is doing processing or not
685. * * When DataTables calculates the column widths to assign to each column, * it finds the longest string in each column and then constructs a * temporary table and reads the widths from that. The problem with this * is that "mmm" is much wider then "iiii", but the latter is a longer * string - thus the calculation can go wrong (doing it properly and putting * it into an DOM object and measuring that is horribly(!) slow). Thus as * a "work around" we provide this option. It will append its value to the * text that is found to be the longest string for the column - i.e. padding. * @type string
686. Table destroyed - nuke any child rows
687. Check to see if there is already a cell (row/colspan) covering our target * insert point. If there is, then there is nothing to do.
688. * * Replace a DataTable which matches the given selector and replace it with * one which has the properties of the new initialisation object passed. If no * table matches the selector, then the new DataTable will be constructed as * per normal. * @type boolean * @default false * * @dtopt Options * @name DataTable.defaults.destroy * * @example * $(document).ready(function() { * $('#example').dataTable({ * "srollY": "200px", * "paginate": false * }); * * // Some time later.... * $('#example').dataTable({ * "filter": false, * "destroy": true * }); * });
689. No filtering, so we want to just use the display master
690. Truncate to the first matched table
691. Empty row
692. Add the ordering method
693. only when `child()` was called with parameters (without it returns an object and this method is not executed)
694. Indicate if DataTables should read DOM data as an object or array Used in _fnGetRowElements
695. * * Reference to internal functions for use by plug-in developers. Note that * these methods are references to internal functions and are considered to be * private. If you use these methods, be aware that they are liable to change * between versions. * @namespace
696. The attribute can be in the format of "#id.class", "#id" or "class" This logic * breaks the string into parts and applies them as needed
697. * * Called when the table has been initialised. Normally DataTables will * initialise sequentially and there will be no need for this function, * however, this does not hold true when using external language information * since that is obtained using an async XHR call. * @type function * @param {object} settings DataTables settings object * @param {object} json The JSON object request from the server - only * present if client-side Ajax sourced data is used * * @dtopt Callbacks * @name DataTable.defaults.initComplete * * @example * $(document).ready(function() { * $('#example').dataTable({ * "initComplete": function(settings, json) { * alert('DataTables has finished its initialisation.'); * } * }); * })
698. Check that the class assignment is correct for sorting
699. * * Calculate the width of columns for the table * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
700. Tell the draw function that we have sorted the data
701. Finished with the table - ditch it
702. Apply the defaults and init options to make a single init object will all options defined from defaults and instance options.
703. Remove the old minimised thead and tfoot elements in the inner table
704. * * Template object for the way in which DataTables holds information about * search information for the global filter and individual column filters. * @namespace
705. * The HTML structure that we want to generate in this function is: * div - scroller * div - scroll head * div - scroll head inner * table - scroll head table * thead - thead * div - scroll body * table - table (master table) * thead - thead clone for sizing * tbody - tbody * div - scroll foot * div - scroll foot inner * table - scroll foot table * tfoot - tfoot
706. * * Sorting enablement flag. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
707. Remove the settings object from the settings array
708. If not given a column array, generate one with nulls
709. * * Array of indexes which are in the current display (after filtering etc) * @type array * @default []
710. else, set the page length
711. Depreciated - remove in 1.11 (providing a plug-in option) Not all sort types have formatting methods, so we have to call their sorting methods.
712. * * This function is called on every 'draw' event, and allows you to * dynamically modify the header row. This can be used to calculate and * display useful information about the table. * @type function * @param {node} head "TR" element for the header * @param {array} data Full table data (as derived from the original HTML) * @param {int} start Index for the current display starting point in the * display array * @param {int} end Index for the current display ending point in the * display array * @param {array int} display Index array to translate the visual position * to the full data array * * @dtopt Callbacks * @name DataTable.defaults.headerCallback * * @example * $(document).ready(function() { * $('#example').dataTable({ * "fheaderCallback": function(head, data, start, end, display) { * head.getElementsByTagName('th')[0].innerHTML = "Displaying "+(end-start)+" records"; * } * }); * })
713. Remove the old striping classes and then add the new one
714. _detailsShown as false or undefined will fall through to return false
715. * * Apply a given function to the display child nodes of an element array (typically * TD children of TR rows * @param {function} fn Method to apply to the objects * @param array {nodes} an1 List of elements to look through for display children * @param array {nodes} an2 Another list (identical structure to the first) - optional * @memberof DataTable#oApi
716. With a capital `D` we return a DataTables API instance rather than a jQuery object
717. If a decimal place other than `.` is used, it needs to be given to the function so we can detect it and replace with a `.` which is the only decimal place Javascript recognises - it is not locale aware.
718. * * Function to get the server-side data. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type function
719. * * Enable or disable ordering on this column. * @type boolean * @default true * * @name DataTable.defaults.column.orderable * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "orderable": false, "targets": [0] } *] }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* { "orderable": false }, * null, * null, * null, * null *] }); * });
720. jslint evil: true, undef: true, browser: true
721. Add the columns
722. Depending on the `data` option for the columns the data can be read to either an object or an array.
723. * * Indicate that if multiple rows are in the header and there is more than * one unique cell per column, if the top one (true) or bottom one (false) * should be used for sorting / title by DataTables. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
724. If there is default sorting required - let's do it. The sort function will do the drawing for us. Otherwise we draw the table regardless of the Ajax source - this allows the table to look initialised for Ajax sourcing data (show 'loading' message possibly)
725. get
726. * * Primary features of DataTables and their enablement state. * @namespace
727. Only apply widths to the DataTables detected header cells - this prevents complex headers from having contradictory sizes applied
728. CommonJS
729. * * Return a function that can be used to get data from a source object, taking * into account the ability to use nested objects as a source * @param {string|int|function} mSource The data source for the object * @returns {function} Data get function * @memberof DataTable#oApi
730. Group the column visibility changes
731. Check if we need to initialise the table (it might not have been handed off to the * language processor)
732. Common renderer - if there is one available for this type use it, otherwise use the default
733. Backwards compatibility for mDataProp
734. Keep the start record on the current page
735. Method extension
736. * * Basically the same as `search`, this parameter defines the individual column * filtering state at initialisation time. The array must be of the same size * as the number of columns, and each element be an object with the parameters * `search` and `escapeRegex` (the latter is optional). 'null' is also * accepted and the default will be used. * @type array * @default [] * * @dtopt Option * @name DataTable.defaults.searchCols * * @example * $(document).ready(function() { * $('#example').dataTable({ * "searchCols": [* null, * { "search": "My filter" }, * null, * { "search": "^[0-9]", "escapeRegex": false } *] * }); * })

737. array of table settings objects
738. There might be colspan cells already in this row, so shift our target * accordingly
739. * * Filtering data cache. This is the same as the cell filtering cache, but * in this case a string rather than an array. This is easily computed with * a join on `_aFilterData`, but is provided as a cache so the join isn't * needed on every search (memory traded for performance) * @type array * @default null * @private
740. * * Enable filtering on the table or not. Note that if this is disabled * then there is no filtering at all on the table, including fnFilter. * To just remove the filtering input use sDom and remove the 'f' option. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
741. * * Define which column(s) an order will occur on for this column. This * allows a column's ordering to take multiple columns into account when * doing a sort or use the data from a different column. For example first * name / last name columns make sense to do a multi-column sort over the * two columns. * @type array|int * @default null <i>Takes the value of the column index automatically</i> * * @name DataTable.defaults.column.orderData * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "orderData": [0, 1], "targets": [0] }, * { "orderData": [1, 0], "targets": [1] }, * { "orderData": 2, "targets": [2] } *] * }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* { "orderData": [0, 1] }, * { "orderData": [1, 0] }, * { "orderData": 2 }, * null, * null *] * }); * });
742. Now do the filter
743. * * Stored plug-in instances * @type object * @default {}
744. End container div
745. Sorting
746. Length
747. Property extension
748. Current page implies that order=current and fitler=applied, since it is fairly senseless otherwise, regardless of what order and search actually are
749. Short cut - selector is a number and no options provided (default is all records, so no need to check if the index is in there, since it must be - dev error if the index doesn't exist).
750. * * Set the ordering for the table. * * @param {array} order 1D array of sorting information to be applied. * @param {array} [...] Optional additional sorting conditions * @returns {DataTables.Api} this
751. * * Enable vertical scrolling. Vertical scrolling will constrain the DataTable * to the given height, and enable scrolling for any data which overflows the * current viewport. This can be used as an alternative to paging to display * a lot of data in a small area (although paging and scrolling can both be * enabled at the same time). This property can be any CSS unit, or a number * (in which case it will be treated as a pixel measurement). * @type string * @default <i>blank string - i.e. disabled</i> * * @dtopt Features * @name DataTable.defaults.scrollY * * @example * $(document).ready(function() { * $('#example').dataTable({ * "scrollY": "200px", * "paginate": false * }); * });
752. Need to add the instance after the instance after the settings object has been added to the settings array, so we can self reference the table instance if more than one
753. Custom filtering
754. * * `Array.prototype` reference. * * @type object * @ignore
755. `iterator` will drop undefined values, but in this case we want them
756. * * Store the applied global search information in case we want to force a * research or compare the old search to a new one. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @namespace * @extends DataTable.models.oSearch
757. Don't destroy the existing prototype, just extend it. Required for jQuery 2's isPlainObject.
758. Negative integer, right to left column counting
759. * * If can be useful to append extra information to the info string at times, * and this variable does exactly that. This information will be appended to * the `info` (`infoEmpty` and `infoFiltered` in whatever combination they are * being used) at all times. * @type string * @default <i>Empty string</i> * * @dtopt Language * @name DataTable.defaults.language.infoPostFix * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "infoPostFix": "All records shown are derived from real information." * } * }); * });
760. Parts are different, return immediately
761. Clone the table header and footer - we can't use the header / footer from the cloned table, since if scrolling is active, the table's real header and footer are contained in different table tags
762. * * Page length change event, fired when number of records to show on each * page (the length) is changed. * @name DataTable#length.dt * @event * @param {event} e jQuery event object * @param {object} o DataTables settings object {@link DataTable.models.oSettings} * @param {integer} len New length
763. * * JSON returned from the server in the last Ajax request * @type object * @default undefined
764. Features
765. Column search objects are in an array, so it needs to be converted element by element
766. * * Pagination plug-in methods. * * Each entry in this object is a function and defines which buttons should * be shown by the pagination rendering method that is used for the table: * {@link DataTable.ext.renderer.pageButton}. The renderer addresses how the * buttons are displayed in the document, while the functions here tell it * what buttons to display. This is done by returning an array of button * descriptions (what each button will do). * * Pagination types (the four built in options and any additional plug-in * options defined here) can be used through the `paginationType` * initialisation parameter. * * The functions defined take two parameters: * * 1. `{int} page` The current page index * 2. `{int} pages` The number of pages in the table * * Each function is expected to return an array where each element of the * array can be one of: * * * `first` - Jump to first page when activated * * `last` - Jump to last page when activated * * `previous` - Show previous page when activated * * `next` - Show next page when activated * * `{int}` - Show page of the index given * * `{array}` - A nested array containing the above elements to add a * containing 'DIV' element (might be useful for styling). * * Note that DataTables v1.9- used this object slightly differently whereby * an object with two functions would be defined for each plug-in. That * ability is still supported by DataTables 1.10+ to provide backwards * compatibility, but this option of use is now decremented and no longer * documented in DataTables 1.10+. * * @type object * @default {} * * @example * // Show previous, next and current page buttons only * $.fn.dataTableExt.oPagination.current = function (page, pages) { * return ['previous', page, 'next']; * };
767. Common actions
768. Build the sort array, with pre-fix and post-fix options if they have been specified
769. Because this approach is destroying and recreating the paging elements, focus is lost on the select button which is bad for accessibility. So we want to restore focus once the draw has completed
770. * * DataTables makes use of renderers when displaying HTML elements for * a table. These renderers can be added or modified by plug-ins to * generate suitable mark-up for a site. For example the Bootstrap * integration plug-in for DataTables uses a paging button renderer to * display pagination buttons in the mark-up required by Bootstrap. * * For further information about the renderers available see * DataTable.ext.renderer * @type string|object * @default null * * @name DataTable.defaults.renderer *
771. Convert to array of TR elements
772. * * Type based plug-ins. * * Each column in DataTables has a type assigned to it, either by automatic * detection or by direct assignment using the `type` option for the column. * The type of a column will effect how it is ordering and search (plug-ins * can also make use of the column type if required). * * @namespace
773. string
774. Hidden header should have zero height, so remove padding and borders. Then set the width based on the real headers
775. * It is useful to have variables which are scoped locally so only the * DataTables functions can access them and they don't leak into global space. * At the same time these functions are often useful over multiple files in the * core and API, so we list, or at least document, all variables which are used * by DataTables as private variables here. This also ensures that there is no * clashing of variable names and that they can easily referenced for reuse.
776. Support for arrays
777. Check it has a unit character already
778. * * Language information for the table. * @namespace * @extends DataTable.defaults.oLanguage
779. * * Remove a row for the table * @param {mixed} target The index of the row from aoData to be deleted, or * the TR element you want to delete * @param {function|null} [callBack] Callback function * @param {bool} [redraw=true] Redraw the table or not * @returns {array} The row that was deleted * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Immediately remove the first row * oTable.fnDeleteRow(0); * });
780. * * Create the HTML header for the table * @param {object} oSettings dataTables settings object * @memberof DataTable#oApi
781. * 2. Take live measurements from the DOM - do not alter the DOM itself!
782. If the number of columns in the DOM equals the number that we have to * process in DataTables, then we can use the offsets that are created by * the web-browser. No custom sizes can be set in order for this to happen, * nor scrolling used
783. * Developer note - See note in model.defaults.js about the use of Hungarian * notation and camel case.

784. callback used for async user interaction
785. * * Class to give to each cell in this column. * @type string * @default <i>Empty string</i> * * @name DataTable.defaults.column.class * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "class": "my_class", "targets": [0] } *] * }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* { "class": "my_class" }, * null, * null, * null, * null *] * }); * });
786. Need to create the HTML if new, or if a rendering function is defined
787. Formatted numbers
788. Resolve any column types that are unknown due to addition or invalidation @todo Can this be moved into a 'data-ready' handler which is called when data is going to be used in the table?
789. * * Developer definable function that is called whenever a cell is created (Ajax source, * etc) or processed for input (DOM source). This can be used as a compliment to mRender * allowing you to modify the DOM element (add background colour for example) when the * element is available. * @type function * @param {element} nTd The TD node that has been created * @param {*} sData The Data for the cell * @param {array|object} oData The data for the whole row * @param {int} iRow The row index for the aoData data store * @default null
790. New container div
791. * * Pagination enabled or not. Note that if this is disabled then length * changing must also be disabled. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
792. Scrolling
793. * * Covert the index of a visible column to the index in the data array (take account * of hidden columns) * @param {object} oSettings dataTables settings object * @param {int} iMatch Visible column index to lookup * @returns {int} i the data index * @memberof DataTable#oApi
794. * * Get an array of unique th elements, one for each column * @param {object} oSettings dataTables settings object * @param {node} nHeader automatically detect the layout from this node - optional * @param {array} aLayout thead/tfoot layout from _fnDetectHeader - optional * @returns array {node} aReturn list of unique th's * @memberof DataTable#oApi
795. Compatibility with 1.9-. In order to read from aaData, check if the default has been changed, if not, check for aaData
796. * 1. Re-create the table inside the scrolling div
797. get row index from id
798. For every cell in the row...
799. Cells
800. * * Get the array indexes of a particular cell from it's DOM element * and column index including hidden columns * @param {node} node this can either be a TR, TD or TH in the table's body * @returns {int} If nNode is given as a TR, then a single index is returned, or * if given as a cell, an array of [row index, column index (visible), * column index (all)] is given. * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * $('#example tbody td').click(function () { * // Get the position of the current data from the node * var aPos = oTable.fnGetPosition(this); * * // Get the data array for this row * var aData = oTable.fnGetData(aPos[0]); * * // Update the data array and return the value * aData[aPos[1]] = 'clicked'; * this.innerHTML = 'clicked'; * }); * * // Init DataTables * oTable = $('#example').dataTable(); * });
801. * * Flag to indicate if the filtering should be case insensitive or not * @type boolean * @default true
802. * * Function used to get a row's id from the row's data * @type function * @default null
803. DataTables settings object
804. If aaSorting is not defined, then we use the first indicator in asSorting in case that has been altered, so the default sort reflects that option
805. * * Enable or disable sorting of columns. Sorting of individual columns can be * disabled by the `sortable` option for each column. * @type boolean * @default true * * @dtopt Features * @name DataTable.defaults.ordering * * @example * $(document).ready(function () { * $('#example').dataTable({ * "ordering": false * }); * });
806. Check if we are dealing with special notation
807. * * Enable or disable automatic column width calculation. This can be disabled * as an optimisation (it takes some time to calculate the widths) if the * tables widths are passed in using `columns`. * @type boolean * @default true * * @dtopt Features * @name DataTable.defaults.autoWidth * * @example * $(document).ready(function () { * $('#example').dataTable({ * "autoWidth": false * }); * });
808. * * Type based ordering. * * The column type tells DataTables what ordering to apply to the table * when a column is sorted upon. The order for each type that is defined, * is defined by the functions available in this object. * * Each ordering option can be described by three properties added to * this object: * * * `{type}-pre` - Pre-formatting function * * `{type}-asc` - Ascending order function * * `{type}-desc` - Descending order function * * All three can be used together, only `{type}-pre` or only * `{type}-asc` and `{type}-desc` together. It is generally recommended * that only `{type}-pre` is used, as this provides the optimal * implementation in terms of speed, although the others are provided * for compatibility with existing Javascript sort functions. * * `{type}-pre`: Functions defined take a single parameter: * * 1. `{*}` Data from the column cell to be prepared for ordering * * And return: * * * `{*}` Data to be sorted upon * * `{type}-asc` and `{type}-desc`: Functions are typical Javascript sort * functions, taking two parameters: * * 1. `{*}` Data to compare to the second parameter * 2. `{*}` Data to compare to the first parameter * * And returning: * * * `{*}` Ordering match: <0 if first parameter should be sorted lower * than the second parameter, ===0 if the two parameters are equal and * >0 if the first parameter should be sorted height than the second * parameter. * * @type object * @default {} * * @example * // Numeric ordering of formatted numbers with a pre-formatter * $.extend($.fn.dataTable.ext.type.order, { * "string-pre": function(x) { * a = (a === "-" || a === "") ? 0 : a.replace(/[^\d\-\.]/g, ""); * return parseFloat(a); * } * }); * * @example * // Case-sensitive string ordering, with no pre-formatting method * $.extend($.fn.dataTable.ext.order, { * "string-case-asc": function(x,y) { * return ((x < y) ? -1 : ((x > y) ? 1 : 0)); * }, * "string-case-desc": function(x,y) { * return ((x < y) ? 1 : ((x > y) ? -1 : 0)); * } * });
809. * * Define the sorting directions that are applied to the column, in sequence * as the column is repeatedly sorted upon - i.e. the first value is used * as the sorting direction when the column if first sorted (clicked on). * Sort it again (click again) and it will move on to the next index. * Repeat until loop. * @type array
810. Simple column / direction passed in
811. Resolve any column types that are unknown due to addition or invalidation @todo As per sort - can this be moved into an event handler?
812. * *
813. DataTables 1.10+ method
814. Selector = function
815. * * Width of the column when it was first "encountered" * @type string * @default null
816. * * Store the applied search for each column - see * {@link DataTable.models.oSearch} for the format that is used for the * filtering information for each column. * @type array * @default []
817. Return an Api.rows() extended instance, so rows().nodes() etc can be used
818. data can be: tr string jQuery or array of any of the above
819. * * Add any control elements for the table - specifically scrolling * @param {object} settings dataTables settings object * @returns {node} Node to add to the DOM * @memberof DataTable#oApi
820. Counting from the right
821. Get all rows
822. Assign the first element to the first item in the instance and truncate the instance and context
823. * * Per cell filtering data cache. As per the sort data cache, used to * increase the performance of the filtering in DataTables * @type array * @default null * @private
824. * * Display or hide the processing indicator * @param {object} settings dataTables settings object * @param {bool} show Show the processing indicator (true) or not (false) * @memberof DataTable#oApi
825. Header and footer callbacks
826. HTML numeric, formatted
827. * * Create a DataTables Api instance, with the currently selected tables for * the Api's context. * @param {boolean} [traditional=false] Set the API instance's context to be * only the table referred to by the `DataTable.ext.iApiIndex` option, as was * used in the API presented by DataTables 1.9- (i.e. the traditional mode), * or if all tables captured in the jQuery object should be used. * @return {DataTables.Api}
828. Single column - sort only on this column
829. Selector - jQuery filtered cells
830. Count from left Count from right (+ because its a negative value)
831. * * Redraw the table * @param {bool} [complete=true] Re-filter and resort (if enabled) the table before the draw. * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Re-draw the table - you wouldn't want to do it here, but it's an

example :-) * oTable.fnDraw(); * });
832. * * If ordering is enabled, then DataTables will perform a first pass sort on * initialisation. You can define which column(s) the sort is performed * upon, and the sorting direction, with this variable. The `sorting` array * should contain an array for each column to be sorted initially containing * the column's index and a direction string ('asc' or 'desc'). * @type array * @default [[0,'asc']] * * @dtopt Option * @name DataTable.defaults.order * * @example * // Sort by 3rd column first, and then 4th column * $(document).ready(function() { * $('#example').dataTable({ * "order": [[2,'asc'], [3,'desc']] * }); * }); * * // No initial sorting * $(document).ready(function() { * $('#example').dataTable({ * "order": [] * }); * });
833. * * If restoring a table - we should restore its striping classes as well * @type array * @default []
834. Provide access to the host jQuery object (circular reference)
835. * * Redraw the table - taking account of the various features which are enabled * @param {object} oSettings dataTables settings object * @param {boolean} [holdPosition] Keep the current paging position. By default * the paging is reset to the first page * @memberof DataTable#oApi
836. * * Set the data property name that DataTables should use to get a row's id * to set as the `id` property in the node. * @type string * @default DT_RowId * * @name DataTable.defaults.rowId
837. Error occurred loading language file, continue on as best we can
838. * * Unique header TH/TD element for this column - this is what the sorting * listener is attached to (if sorting is enabled.) * @type node * @default null
839. Try to get width information from the DOM. We can't get it from CSS as we'd need to parse the CSS stylesheet. `width` option can override
840. Visible index given, convert to column index
841. * * Display information string for when the table is empty. Typically the * format of this string should match `info`. * @type string * @default Showing 0 to 0 of 0 entries * * @dtopt Language * @name DataTable.defaults.language.infoEmpty * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "infoEmpty": "No entries to show" * } * }); * });
842. Plug-in features
843. * * Filter the input based on data * @param {string} sInput String to filter the table on * @param {int|null} [iColumn] Column to limit filtering to * @param {bool} [bRegex=false] Treat as regular expression or not * @param {bool} [bSmart=true] Perform smart filtering or not * @param {bool} [bShowGlobal=true] Show the input global filter in it's input box(es) * @param {bool} [bCaseInsensitive=true] Do case-insensitive matching (true) or not (false) * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Sometime later - filter... * oTable.fnFilter('test string'); * });
844. * * Table information element (the 'Showing x of y records' div) enable * flag. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
845. this has same length as context - one entry for each table
846. Allow plug-ins and external processes to modify the data
847. Remove row stripe classes if they are already on the table row
848. Value
849. consider. See jsperf.com/compare-array-unique-versions/4 for more information.
850. * * This property can be used to read data from any data source property, * including deeply nested objects / properties. `data` can be given in a * number of different ways which effect its behaviour: * * * `integer` - treated as an array index for the data source. This is the * default that DataTables uses (incrementally increased for each column). * * `string` - read an object property from the data source. There are * three 'special' options that can be used in the string to alter how * DataTables reads the data from the source object: * * `.` - Dotted Javascript notation. Just as you use a `.` in * Javascript to read from nested objects, so to can the options * specified in `data`. For example: `browser.version` or * `browser.name`. If your object parameter name contains a period, use * `\\` to escape it - i.e. `first\\.name`. * * `[]` - Array notation. DataTables can automatically combine data * from and array source, joining the data with the characters provided * between the two brackets. For example: `name[, ]` would provide a * comma-space separated list from the source array. If no characters * are provided between the brackets, the original array source is * returned. * * `()` - Function notation. Adding `()` to the end of a parameter will * execute a function of the name given. For example: `browser()` for a * simple function on the data source, `browser.version()` for a * function in a nested property or even `browser().version` to get an * object property if the function called returns an object. Note that * function notation is recommended for use in `render` rather than * `data` as it is much simpler to use as a renderer. * * `null` - use the original data source for the row rather than plucking * data directly from it. This action has effects on two other * initialisation options: * * `defaultContent` - When null is given as the `data` option and * `defaultContent` is specified for the column, the value defined by * `defaultContent` will be used for the cell. * * `render` - When null is used for the `data` option and the `render` * option is specified for the column, the whole data source for the * row is used for the renderer. * * `function` - the function given will be executed whenever DataTables * needs to set or get the data for a cell in the column. The function * takes three parameters: * * Parameters: * * `{array|object}` The data source for the row * * `{string}` The type call data requested - this will be 'set' when * setting data or 'filter', 'display', 'type', 'sort' or undefined * when gathering data. Note that when `undefined` is given for the * type DataTables expects to get the raw data for the object back< * * `{*}` Data to set when the second parameter is 'set'. * * Return: * * The return value from the function is not required when 'set' is * the type of call, but otherwise the return is what will be used * for the data requested. * * Note that `data` is a getter and setter option. If you just require * formatting of data for output, you will likely want to use `render` which * is simply a getter and thus simpler to use. * * Note that prior to DataTables 1.9.2 `data` was called `mDataProp`. The * name change reflects the flexibility of this property and is consistent * with the naming of mRender. If 'mDataProp' is given, then it will still * be used by DataTables, as it automatically maps the old name to the new * if required. * * @type string|int|function|null * @default null <i>Use automatically calculated column index</i> * * @name DataTable.defaults.column.data * @dtopt Columns * * @example * // Read table data from objects * // JSON structure for each row: * // { * // "engine": {value}, * // "browser": {value}, * // "platform": {value}, * // "version": {value}, * // "grade": {value} * // } * $(document).ready(function() { * $('#example').dataTable({ * "ajaxSource": "sources/objects.txt", * "columns": [* { "data": "engine" }, * { "data": "browser" }, * { "data": "platform" }, * { "data": "version" }, * { "data": "grade" } *] * }); * }); * * @example * // Read information from deeply nested objects * // JSON structure for each row: * // { * // "engine": {value}, * // "browser": {value}, * // "platform": { * // "inner": {value} * // }, * // "details": [* // {value}, {value} * //] * // } * $(document).ready(function() { * $('#example').dataTable({ * "ajaxSource": "sources/deep.txt", * "columns": [* { "data": "engine" }, * { "data": "browser" }, * { "data": "platform.inner" }, * { "data": "platform.details.0" }, * { "data": "platform.details.1" } *] * }); * }); * * @example * // Using `data` as a function to provide different information for * // sorting, filtering and display. In this case, currency (price) * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [{ * "targets": [0], * "data": function (source, type, val) { * if (type === 'set') { * source.price = val; * // Store the computed dislay and filter values for efficiency * source.price_display = val=="" ? "" : "$"+numberFormat(val); * source.price_filter = val=="" ? "" : "$"+numberFormat(val)+" "+val; * return; * } * else if (type === 'display') { * return source.price_display; * } * else if (type === 'filter') { * return source.price_filter; * } * // 'sort', 'type' and undefined all just use the integer * return source.price; * } * }] * }); * }); * * @example * // Using default content * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [{ * "targets": [0], * "data": null, * "defaultContent": "Click to edit" * }] * }); * }); * * @example * // Using array notation - outputting a list from an array * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [{ * "targets": [0], * "data": "name[,]" * }] * }); * }); *
851. If processing is enabled use a timeout to allow the processing display to be shown - otherwise to it synchronously
852. * * Convert from camel case parameters to Hungarian notation. This is made public * for the extensions to provide the same ability as DataTables core to accept * either the 1.9 style Hungarian notation, or the 1.10+ style camelCase * parameters. * * @param {object} src The model object which holds all parameters that can be * mapped. * @param {object} user The object to convert from camel case to Hungarian. * @param {boolean} force When set to `true`, properties which already have a * Hungarian value in the `user` object will be overwritten. Otherwise they * won't be.
853. For testing and plug-ins to use
854. * * Array referencing the nodes which are used for the features. The * parameters of this object match what is allowed by sDom - i.e. * <ul> * <li>'l' - Length changing</li> * <li>'f' - Filtering input</li> * <li>'t' - The table!</li> * <li>'i' - Information</li> * <li>'p' - Pagination</li> * <li>'r' - pRocessing</li> * </ul> * @type array * @default []
855. * * Internal functions, exposed for used in plug-ins. * * Please note that you should not need to use the internal methods for * anything other than a plug-in (and even then, try to avoid if possible). * The internal function may change between releases. * * @type object * @default {}
856. Legacy aliases
857. * * Calculate the 'type' of a column * @param {object} settings dataTables settings object * @memberof DataTable#oApi
858. Can't use `select` variable as user might provide their own and the reference is broken by the use of outerHTML
859. * Rows * * {} - no selector - use all available rows * {integer} - row aoData index * {node} - TR node * {string} - jQuery selector to apply to the TR elements * {array} - jQuery array of nodes, or simply an array of TR nodes *
860. * Columns * * {integer} - column index (>=0 count from left, <0 count from right) * "{integer}:visIdx" - visible column index (i.e. translate to column index) (>=0 count from left, <0 count from right) * "{integer}:visible" - alias for {integer}:visIdx (>=0 count from left, <0 count from right) * "{string}:name" - column name * "{string}" - jQuery selector on column header nodes *

861. * * Flag to indicate if HTML5 data attributes should be used as the data * source for filtering or sorting. True is either are. * @type boolean * @default false * @private
862. * * Set the Ajax URL. Note that this will set the URL for all tables in the * current context. * * @param {string} url URL to set. * @returns {DataTables.Api} this
863. Feature sorting overrides column specific when off
864. * * A list of the columns that sorting should occur on when this column * is sorted. That this property is an array allows multi-column sorting * to be defined for a column (for example first name / last name columns * would benefit from this). The values are integers pointing to the * columns to be sorted on (typically it will be a single integer pointing * at itself, but that doesn't need to be the case). * @type array
865. * * This function is called when a TR element is created (and all TD child * elements have been inserted), or registered if using a DOM source, allowing * manipulation of the TR element (adding classes etc). * @type function * @param {node} row "TR" element for the current row * @param {array} data Raw data array for this row * @param {int} dataIndex The index of this row in the internal aoData array * * @dtopt Callbacks * @name DataTable.defaults.createdRow * * @example * $(document).ready(function() { * $('#example').dataTable({ * "createdRow": function(row, data, dataIndex) { * // Bold the grade for all 'A' grade browsers * if (data[4] == "A") * { * $('td:eq(4)', row).html('<b>A</b>'); * } * } * }); * });
866. * * This property is the rendering partner to `data` and it is suggested that * when you want to manipulate data for display (including filtering, * sorting etc) without altering the underlying data for the table, use this * property. `render` can be considered to be the the read only companion to * `data` which is read / write (then as such more complex). Like `data` * this option can be given in a number of different ways to effect its * behaviour: * * * `integer` - treated as an array index for the data source. This is the * default that DataTables uses (incrementally increased for each column). * * `string` - read an object property from the data source. There are * three 'special' options that can be used in the string to alter how * DataTables reads the data from the source object: * * `.` - Dotted Javascript notation. Just as you use a `.` in * Javascript to read from nested objects, so to can the options * specified in `data`. For example: `browser.version` or * `browser.name`. If your object parameter name contains a period, use * `\\` to escape it - i.e. `first\\.name`. * * `[]` - Array notation. DataTables can automatically combine data * from and array source, joining the data with the characters provided * between the two brackets. For example: `name[, ]` would provide a * comma-space separated list from the source array. If no characters * are provided between the brackets, the original array source is * returned. * * `()` - Function notation. Adding `()` to the end of a parameter will * execute a function of the name given. For example: `browser()` for a * simple function on the data source, `browser.version()` for a * function in a nested property or even `browser().version` to get an * object property if the function called returns an object. * * `object` - use different data for the different data types requested by * DataTables ('filter', 'display', 'type' or 'sort'). The property names * of the object is the data type the property refers to and the value can * defined using an integer, string or function using the same rules as * `render` normally does. Note that an `_` option _must_ be specified. * This is the default value to use if you haven't specified a value for * the data type requested by DataTables. * * `function` - the function given will be executed whenever DataTables * needs to set or get the data for a cell in the column. The function * takes three parameters: * * Parameters: * * {array|object} The data source for the row (based on `data`) * * {string} The type call data requested - this will be 'filter', * 'display', 'type' or 'sort'. * * {array|object} The full data source for the row (not based on * `data`) * * Return: * * The return value from the function is what will be used for the * data requested. * * @type string|int|function|object|null * @default null Use the data source value. * * @name DataTable.defaults.column.render * @dtopt Columns * * @example * // Create a comma separated list from an array of objects * $(document).ready(function() { * $('#example').dataTable({ * "ajaxSource": "sources/deep.txt", * "columns": [* { "data": "engine" }, * { "data": "browser" }, * { * "data": "platform", * "render": "[,].name" * } *] * }); * }); * * @example * // Execute a function to obtain data * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [{ * "targets": [0], * "data": null, // Use the full data source object for the renderer's source * "render": "browserName()" * }] * }); * }); * * @example * // As an object, extracting different data for the different types * // This would be used with a data source such as: * // { "phone": 5552368, "phone_filter": "5552368 555-2368", "phone_display": "555-2368" } * // Here the `phone` integer is used for sorting and type detection, while `phone_filter` * // (which has both forms) is used for filtering for if a user inputs either format, while * // the formatted phone number is the one that is shown in the table. * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [{ * "targets": [0], * "data": null, // Use the full data source object for the renderer's source * "render": { * "_": "phone", * "filter": "phone_filter", * "display": "phone_display" * } * }] * }); * }); * * @example * // Use as a function to create a link from the data source * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [{ * "targets": [0], * "data": "download_link", * "render": function (data, type, full) { * return '<a href="'+data+'">Download</a>'; * } * }] * }); * });
867. * * Dictate the positioning of DataTables' control elements - see * {@link DataTable.model.oInit.sDom}. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type string * @default null
868. Style attribute
869. Search
870. Selector - none
871. Blitz all `DT` namespaced events (these are internal events, the lowercase, `dt` events are user subscribed and they are responsible for removing them
872. Restore key features - todo - for 1.11 this needs to be done by subscribed events
873. * * Set the value for a specific cell, into the internal data cache * @param {object} settings dataTables settings object * @param {int} rowIdx aoData row id * @param {int} colIdx Column index * @param {*} val Value to set * @memberof DataTable#oApi
874. Make a copy of the master layout array, but without the visible columns in it
875. Check for an 'overflow' they case for displaying the table
876. * * DataTables settings object - this holds all the information needed for a * given table, including configuration, data and current application of the * table options. DataTables does not have a single instance for each DataTable * with the settings attached to that instance, but rather instances of the * DataTable "class" are created on-the-fly as needed (typically by a * $().dataTable() call) and the settings object is then applied to that * instance. * * Note that this object is related to {@link DataTable.defaults} but this * one is the internal data store for DataTables's cache of columns. It should * NOT be manipulated outside of DataTables. Any configuration should be done * through the initialisation options. * @namespace * @todo Really should attach the settings object to individual instances so we * don't need to create new instances on each $().dataTable() call (if the * table already exists). It would also save passing oSettings around and * into every single function. However, this is a very significant * architecture change for DataTables and will almost certainly break * backwards compatibility with older installations. This is something that * will be done in 2.0.
877. Want argument shifting here and in __row_selector?
878. So the array reference doesn't break set the results into the existing array
879. * State API methods
880. Deal with the footer - add classes if required
881. Must update the applied array over the rows for the columns
882. Get the language definitions from a file - because this Ajax call makes the language * get async to the remainder of this function we use bInitHandedOff to indicate that * _fnInitialise will be fired by the returned Ajax handler, rather than the constructor
883. * * The class to apply to all TD elements in the table's TBODY for the column * @type string * @default null
884. When the body is scrolled, then we also want to scroll the headers
885. * * Cache the table ID for quick access * @type string * @default <i>Empty string</i>
886. Convert any user input sizes into pixel sizes
887. * * Get the number of records in the current record set, before filtering * @type function
888. Update display on each draw
889. * * Allows a default value to be given for a column's data, and will be used * whenever a null data source is encountered (this can be because mData * is set to null, or because the data source itself is null). * @type string * @default null
890. * * Send the XHR HTTP method - GET or POST (could be PUT or DELETE if * required). * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type string
891. * * Get the settings for a particular table for external manipulation * @returns {object} DataTables settings object. See * {@link DataTable.models.oSettings} * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * var oSettings = oTable.fnSettings(); * * // Show an example parameter from the settings * alert(oSettings._iDisplayStart); * });
892. * * Update the information elements in the display * @param {object} settings dataTables settings object * @memberof DataTable#oApi
893. The inner call to setData has already traversed through the remainder of the source and has set the data, thus we can exit here
894. Returned item is the API instance that was passed in, return it
895. Get the data to sort a column, be it from cache, fresh (populating the cache), or from a sort formatter
896. Adjust the position of the header in case we loose the y-scrollbar
897. In server-side processing all filtering is done by the server, so no point hanging around here
898. adjust column sizing will call this function again
899. If array notation is used, we just want to strip it and use the property name and assign the value. If it isn't used, then we get the result we want anyway

900. selector extensions
901. Show information about the table
902. * * Class to be applied to the header element when sorting on this column * @type string * @default null
903. * * Store data information - see {@link DataTable.models.oRow} for detailed * information. * @type array * @default []
904. * * Callbacks for operating on the settings object once the saved state has been * loaded * @type array * @default []
905. Reset the init display for cookie saving. We've already done a filter, and therefore cleared it before. So we need to make it appear 'fresh'
906. If the length menu is given, but the init display length is not, use the length menu
907. i18n method for extensions to be able to use the language object from the DataTable
908. * * State saving enablement flag. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type boolean
909. * * Cache of the class name that DataTables has applied to the row, so we * can quickly look at this variable rather than needing to do a DOM check * on className for the nTr property. * @type string * @default <i>Empty string</i> * @private
910. * * Version string for plug-ins to check compatibility. Allowed format is * `a.b.c-d` where: a:int, b:int, c:int, d:string(dev|beta|alpha). `d` is used * only for non-release builds. See http://semver.org/ for more information. * @member * @type string * @default Version number
911. Protect against out of sequence returns
912. Read all widths in next pass
913. Table has been built, attach to the document so we can work with it. A holding element is used, positioned at the top of the container with minimal height, so it has no effect on if the container scrolls or not. Otherwise it might trigger scrolling when it actually isn't needed
914. * * Number of rows to display on a single page when using pagination. If * feature enabled (`lengthChange`) then the end user will be able to override * this to a custom setting using a pop-up menu. * @type int * @default 10 * * @dtopt Options * @name DataTable.defaults.pageLength * * @example * $(document).ready(function() { * $('#example').dataTable({ * "pageLength": 50 * }); * })
915. Apply the column definitions
916. Backwards compatibility, before we apply all the defaults
917. Fire off the destroy callbacks for plug-ins etc
918. Restore the width of the original table - was read from the style property, so we can restore directly to that
919. * * Enable or disable filtering on the data in this column. * @type boolean * @default true * * @name DataTable.defaults.column.searchable * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { "searchable": false, "targets": [0] } *] }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* { "searchable": false }, * null, * null, * null, * null *] }); * });
920. For objects, we need to buzz down into the object to copy parameters
921. value modified by the draw
922. * * Classes to use for the striping of a table. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type array * @default []
923. Condition allows simply [] to be passed in
924. * * Bind an event handers to allow a click or return key to activate the callback. * This is good for accessibility since a return on the keyboard will have the * same effect as a click, if the element has focus. * @param {element} n Element to bind the action to * @param {object} oData Data object to pass to the triggered function * @param {function} fn Callback function for when the event is triggered * @memberof DataTable#oApi
925. Order by the selected column(s)
926. Use the default column options function to initialise classes etc
927. Scrolling feature / quirks detection
928. Must be done after everything which can be overridden by the state saving!
929. * * Create an Ajax call based on the table's settings, taking into account that * parameters can have multiple forms, and backwards compatibility. * * @param {object} oSettings dataTables settings object * @param {array} data Data to send to the server, required by * DataTables - may be augmented by developer callbacks * @param {function} fn Callback function to run when data is obtained
930. * * Log an error message * @param {object} settings dataTables settings object * @param {int} level log error messages, or display them to the user * @param {string} msg error message * @param {int} tn Technical note id to get more information about the error. * @memberof DataTable#oApi
931. * * Which type of pagination should be used. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type string * @default two_button
932. Add a draw callback for the pagination on first instance, to update the paging display
933. * * Callback that is called when the state has been loaded from the state saving method * and the DataTables settings object has been modified as a result of the loaded state. * @type function * @param {object} settings DataTables settings object * @param {object} data The state object that was loaded * * @dtopt Callbacks * @name DataTable.defaults.stateLoaded * * @example * // Show an alert with the filtering value that was saved * $(document).ready(function() { * $('#example').dataTable({ * "stateSave": true, * "stateLoaded": function (settings, data) { * alert('Saved filter was: '+data.oSearch.sSearch); * } * }); * });
934. Non-API return - just fire it back
935. Compatibility with 1.9-.
936. Backwards compatibility with the bEscapeRegex option
937. * * Detect the data source being used for the table. Used to simplify the code * a little (ajax) and to make it compress a little smaller. * * @param {object} settings dataTables settings object * @returns {string} Data source * @memberof DataTable#oApi
938. * Final init * Cache the header, body and footer as required, creating them if needed
939. * * Add a column to the list used for the table with default values * @param {object} oSettings dataTables settings object * @param {node} nTh The th element for this column * @memberof DataTable#oApi
940. * * Data submitted as part of the last Ajax request * @type object * @default undefined
941. * * This string gives information to the end user about the information * that is current on display on the page. The following tokens can be * used in the string and will be dynamically replaced as the table * display updates. This tokens can be placed anywhere in the string, or * removed as needed by the language requires: * * * `\_START\_` - Display index of the first record on the current page * * `\_END\_` - Display index of the last record on the current page * * `\_TOTAL\_` - Number of records in the table after filtering * * `\_MAX\_` - Number of records in the table without filtering * * `\_PAGE\_` - Current page number * * `\_PAGES\_` - Total number of pages of data in the table * * @type string * @default Showing _START_ to _END_ of _TOTAL_ entries * * @dtopt Language * @name DataTable.defaults.language.info * * @example * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "info": "Showing page _PAGE_ of _PAGES_" * } * }); * });
942. * * Initialisation object that is used for the table * @type object * @default null
943. Give an empty string for rendering / sorting etc
944. When infinite scrolling, we are always starting at 1. _iDisplayStart is used only internally
945. string-asc and -desc are retained only for compatibility with the old sort methods
946. "Hide" the header and footer that we used for the sizing. We need to keep the content of the cell so that the width applied to the header and body both match, but we want to hide it completely. We want to also fix their width to what they currently are
947. Misc
948. * Sorting * @todo For modularisation (1.11) this needs to do into a sort start up handler
949. Parts are the same, keep comparing
950. * * DIV container for the footer scrolling table if scrolling
951. * * The exact opposite of 'opening' a row, this function will close any rows which * are currently 'open'. * @param {node} nTr the table row to 'close' * @returns {int} 0 on success, or 1 if failed (can't find the row) * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable; * * // 'open' an information row when a row is clicked on * $('#example tbody tr').click(function () { * if (oTable.fnIsOpen(this)) { * oTable.fnClose(this); * } else { * oTable.fnOpen(this, "Temporary row opened", "info_row"); * } * }); * * oTable = $('#example').dataTable(); * });
952. * * Create a mapping object that allows camel case parameters to be looked up * for their Hungarian counterparts. The mapping is stored in a private * parameter called `_hungarianMap` which can be accessed on the source object. * @param {object} o * @memberof DataTable#oApi
953. Initialisation complete - table can be drawn
954. * * Enable horizontal scrolling. When a table is too wide to fit into a * certain layout, or you have a large number of columns in the table, you * can enable x-scrolling to show the table in a viewport, which can be * scrolled. This property can be `true` which will allow the table to * scroll horizontally when needed, or

any CSS unit, or a number (in which * case it will be treated as a pixel measurement). Setting as simply `true` * is recommended. * @type boolean|string * @default <i>blank string - i.e. disabled</i> * * @dtopt Features * @name DataTable.defaults.scrollX * * @example * $(document).ready(function() { * $('#example').dataTable({ * "scrollX": true, * "scrollCollapse": true * }); * });
955. Compatibility with 1.9-, allow fnServerData and event to manipulate
956. Update all other filter input elements for the new display
957. * * Duration for which the saved state information is considered valid. After this period * has elapsed the state will be returned to the default. * Value is given in seconds. * @type int * @default 7200 <i>(2 hours)</i> * * @dtopt Options * @name DataTable.defaults.stateDuration * * @example * $(document).ready(function() { * $('#example').dataTable({ * "stateDuration": 60*60*24; // 1 day * }); * })
958. * * Callback functions for the header on each draw. * @type array * @default []
959. pRocessing
960. * * Allows the end user to select the size of a formatted page from a select * menu (sizes are 10, 25, 50 and 100). Requires pagination (`paginate`). * @type boolean * @default true * * @dtopt Features * @name DataTable.defaults.lengthChange * * @example * $(document).ready(function () { * $('#example').dataTable({ * "lengthChange": false * }); * });
961. Strings for the method names to help minification
962. columns and rows share the same structure. 'this' is an array of column indexes for each context
963. * * Enable or disable filtering of data. Filtering in DataTables is "smart" in * that it allows the end user to input multiple words (space separated) and * will match a row containing those words, even if not in the order that was * specified (this allow matching across multiple columns). Note that if you * wish to use filtering in DataTables this must remain 'true' - to remove the * default filtering input box and retain filtering abilities, please use * {@link DataTable.defaults.dom}. * @type boolean * @default true * * @dtopt Features * @name DataTable.defaults.searching * * @example * $(document).ready(function () { * $('#example').dataTable({ * "searching": false * }); * });
964. * * This function will make DataTables recalculate the column sizes, based on the data * contained in the table and the sizes applied to the columns (in the DOM, CSS or * through the sWidth parameter). This can be useful when the width of the table's * parent element changes (for example a window resize). * @param {boolean} [bRedraw=true] Redraw the table or not, you will typically want to * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable({ * "sScrollY": "200px", * "bPaginate": false * }); * * $(window).on('resize', function () { * oTable.fnAdjustColumnSizing(); * }); * });
965. Width for each column to use
966. Convert the macros
967. Invalidate the type for a specific column (if given) or all columns since the data might have changed
968. Visibility - add or remove as required
969. * * Partner property to mData which is used (only when defined) to get * the data - i.e. it is basically the same as mData, but without the * 'set' option, and also the data fed to it is the result from mData. * This is the rendering method to match the data method of mData. * @type function|int|string|null * @default null
970. * * Take a TR element and convert it to an index in aoData * @param {object} oSettings dataTables settings object * @param {node} n the TR element to find * @returns {int} index if the node is found, null if not * @memberof DataTable#oApi
971. * * Escape a string such that it can be used in a regular expression * @param {string} sVal string to escape * @returns {string} escaped string * @memberof DataTable#oApi
972. Are we already doing some kind of sort on this column?
973. Ensure the table has an ID - required for accessibility
974. Row callback functions - might want to manipulate the row iRowCount and j are not currently documented. Are they at all useful?
975. * * Enable or disable the table information display. This shows information * about the data that is currently visible on the page, including information * about filtered data if that action is being performed. * @type boolean * @default true * * @dtopt Features * @name DataTable.defaults.info * * @example * $(document).ready(function () { * $('#example').dataTable({ * "info": false * }); * });
976. * * The `columns` option in the initialisation parameter allows you to define * details about the way individual columns behave. For a full list of * column options that can be set, please see * {@link DataTable.defaults.column}. Note that if you use `columns` to * define your columns, you must have an entry in the array for every single * column that you have in your table (these can be null if you don't which * to specify any options). * @member * * @name DataTable.defaults.column
977. Order
978. * * Configure DataTables to use server-side processing. Note that the * `ajax` parameter must also be given in order to give DataTables a * source to obtain the required data for each draw. * @type boolean * @default false * * @dtopt Features * @dtopt Server-side * @name DataTable.defaults.serverSide * * @example * $(document).ready(function () { * $('#example').dataTable({ * "serverSide": true, * "ajax": "xhr.php" * }); * });
979. * * Browser scrollbar width * @type integer * @default 0
980. * * Array of callback functions for state saving. Each array element is an * object with the following parameters: * <ul> * <li>function:fn - function to call. Takes two parameters, oSettings * and the JSON string to save that has been thus far created. Returns * a JSON string to be inserted into a json object * (i.e. '"param": [0, 1, 2]')</li> * <li>string:sName - name of callback</li> * </ul> * @type array * @default []
981. Filtering
982. Recalculate the sanity width
983. Restore for next time around
984. Same as a reload, but makes sense to present it for easy access after a url change
985. * * Unique DataTables instance counter * * @type int * @private
986. Note that we can't call this function 'length()' because `length` is a Javascript property of functions which defines how many arguments the function expects.
987. * * Context selector for the API's context (i.e. the tables the API instance * refers to. * * @name DataTable.Api#tables * @param {string|integer} [selector] Selector to pick which tables the iterator * should operate on. If not given, all tables in the current context are * used. This can be given as a jQuery selector (for example `':gt(0)'`) to * select multiple tables or as an integer to select a single table. * @returns {DataTable.Api} Returns a new API instance if a selector is given.
988. Reading from data object, update the DOM
989. * * Filter the data table based on user input and draw the table * @param {object} settings dataTables settings object * @param {string} input string to filter on * @param {int} force optional - force a research of the master array (1) or not (undefined or 0) * @param {bool} regex treat as a regular expression or not * @param {bool} smart perform smart filtering or not * @param {bool} caseInsensitive Do case insenstive matching or not * @memberof DataTable#oApi
990. * * Get the data submitted in the last Ajax request
991. iDataSort to be applied (backwards compatibility), but aDataSort will take * priority if defined
992. Expand the cell to cover as many columns as needed
993. Store the interesting variables
994. Expand the cell to cover as many rows as needed
995. No sorting required if server-side or no sorting array
996. * * Register a callback function. Easily allows a callback function to be added to * an array store of callback functions that can then all be called together. * @param {object} oSettings dataTables settings object * @param {string} sStore Name of the array storage for the callbacks in oSettings * @param {function} fn Function to be called back * @param {string} sName Identifying name for the callback (i.e. a label) * @memberof DataTable#oApi
997. * * Convert from the internal Hungarian notation to camelCase for external * interaction * @param {object} obj Object to convert * @returns {object} Inverted object * @memberof DataTable#oApi
998. * * State that was loaded. Useful for back reference * @type object * @default null
999. * * Map one parameter onto another * @param {object} o Object to map * @param {*} knew The new parameter name * @param {*} old The old parameter name
1000. * * Return a function that can be used to set data from a source object, taking * into account the ability to use nested objects as a source * @param {string|int|function} mSource The data source for the object * @returns {function} Data set function * @memberof DataTable#oApi
1001. If the input is blank - we just want the full data set
1002. * * tabindex attribute value that is added to DataTables control elements, allowing * keyboard navigation of the table and its controls.
1003. * * Take a TD element and convert it into a column data index (not the visible index) * @param {object} oSettings dataTables settings object * @param {int} iRow The row number the TD/TH can be found in * @param {node} n The TD/TH element to find * @returns {int} index if the node is found, -1 if not * @memberof DataTable#oApi
1004. DataTables 1.9- compatible method

1005. If we have space to show extra rows (backing up from the end point - then do so
1006. * * Flag to indicate if DataTables is to use its smart filtering or not. * @type boolean * @default true
1007. * * Provide a common method for plug-ins to check the version of DataTables being * used, in order to ensure compatibility. * * @param {string} version Version string to check for, in the format "X.Y.Z". * Note that the formats "X" and "X.Y" are also acceptable. * @returns {boolean} true if this version of DataTables is greater or equal to * the required version, or false if this version of DataTales is not * suitable * @static * @dtopt API-Static * * @example * alert($.fn.dataTable.versionCheck('1.9.0'));
1008. * * Indicate if the browser incorrectly calculates width:100% inside a * scrolling element (IE6/7) * @type boolean * @default false
1009. * * DataTables API class - used to control and interface with one or more * DataTables enhanced tables. * * The API class is heavily based on jQuery, presenting a chainable interface * that you can use to interact with tables. Each instance of the API class has * a "context" - i.e. the tables that it will operate on. This could be a single * table, all tables on a page or a sub-set thereof. * * Additionally the API is designed to allow you to easily work with the data in * the tables, retrieving and manipulating it as required. This is done by * presenting the API class as an array like interface. The contents of the * array depend upon the actions requested by each method (for example * `rows().nodes()` will return an array of nodes, while `rows().data()` will * return an array of objects or arrays depending upon your table's * configuration). The API object has a number of array like methods (`push`, * `pop`, `reverse` etc) as well as additional helper methods (`each`, `pluck`, * `unique` etc) to assist your working with the data held in a table. * * Most methods (those which return an Api instance) are chainable, which means * the return from a method call also has all of the methods available that the * top level object had. For example, these two calls are equivalent: * * // Not chained * api.row.add({...}); * api.draw(); * * // Chained * api.row.add({...}).draw(); * * @class DataTable.Api * @param {array|object|string|jQuery} context DataTable identifier. This is * used to define which DataTables enhanced tables this API will operate on. * Can be one of: * * * `string` - jQuery selector. Any DataTables' matching the given selector * with be found and used. * * `node` - `TABLE` node which has already been formed into a DataTable. * * `jQuery` - A jQuery object of `TABLE` nodes. * * `object` - DataTables settings object * @param {array} [data] Data to initialise the Api instance with. * * @example * // Direct initialisation during DataTables construction * var api = $('#example').DataTable(); * * @example * // Initialisation using a DataTables jQuery object * var api = $('#example').dataTable().api(); * * @example * // Initialisation as a constructor * var api = new $.fn.DataTable.Api('table.dataTable');
1010. * * Almost identical to $ in operation, but in this case returns the data for the matched * rows - as such, the jQuery selector used should match TR row nodes or TD/TH cell nodes * rather than any descendants, so the data can be obtained for the row/cell. If matching * rows are found, the data returned is the original data array/object that was used to * create the row (or a generated array if from a DOM source). * * This method is often useful in-combination with $ where both functions are given the * same parameters and the array indexes will match identically. * @param {string|node|jQuery} sSelector jQuery selector or node collection to act on * @param {object} [oOpts] Optional parameters for modifying the rows to be included * @param {string} [oOpts.filter=none] Select elements that meet the current filter * criterion ("applied") or all elements (i.e. no filter). * @param {string} [oOpts.order=current] Order of the data in the processed array. * Can be either 'current', whereby the current sorting of the table is used, or * 'original' whereby the original order the data was read into the table is used. * @param {string} [oOpts.page=all] Limit the selection to the currently displayed page * ("current") or not ("all"). If 'current' is given, then order is assumed to be * 'current' and filter is 'applied', regardless of what they might be given as. * @returns {array} Data for the matched elements. If any elements, as a result of the * selector, were not TR, TD or TH elements in the DataTable, they will have a null * entry in the array. * @dtopt API * @deprecated Since v1.10 * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Get the data from the first row in the table * var data = oTable._('tr:first'); * * // Do something useful with the data * alert("First cell is: "+data[0]); * }); * * @example * $(document).ready(function() { * var oTable = $('#example').dataTable(); * * // Filter to 'Webkit' and get all data for * oTable.fnFilter('Webkit'); * var data = oTable._('tr', {"search": "applied"}); * * // Do something with the data * alert(data.length+" rows matched the search"); * });
1011. * * DataTables build type (expanded by the download builder) * * @type string
1012. Holding element for speed
1013. * * Create a wrapper function for exporting an internal functions to an external API. * @param {string} fn API function name * @returns {function} wrapped function * @memberof DataTable#internal
1014. If there is a width attr, we want to attach an event listener which allows the table sizing to automatically adjust when the window is resized. Use the width attr rather than CSS, since we can't know if the CSS is a relative value or absolute - DOM read is always px.
1015. [{ name: 'data' -- string - Property name val: function () {}, -- function - Api method (or undefined if just an object methodExt: [...], -- array - Array of Api object definitions to extend the method result propExt: [...] -- array - Array of Api object definitions to extend the property }, { name: 'row' val: {}, methodExt: [...], propExt: [{ name: 'data' val: function () {}, methodExt: [...], propExt: [...] }, ...] }]
1016. * * Take an array of integers (index array) and remove a target integer (value - not * the key!) * @param {array} a Index array to target * @param {int} iTarget value to find * @memberof DataTable#oApi
1017. * * Column sorting and filtering type * @type string * @default null
1018. Size the table as a whole
1019. * * Add a single new row or multiple rows of data to the table. Please note * that this is suitable for client-side processing only - if you are using * server-side processing (i.e. "bServerSide": true), then to add data, you * must add it to the data source, i.e. the server-side, through an Ajax call. * @param {array|object} data The data to be added to the table. This can be: * <ul> * <li>1D array of data - add a single row with the data provided</li> * <li>2D array of arrays - add multiple rows in a single call</li> * <li>object - data object when using <i>mData</i></li> * <li>array of objects - multiple data objects when using <i>mData</i></li> * </ul> * @param {bool} [redraw=true] redraw the table or not * @returns {array} An array of integers, representing the list of indexes in * <i>aoData</i> ({@link DataTable.models.oSettings}) that have been added to * the table. * @dtopt API * @deprecated Since v1.10 * * @example * // Global var for counter * var giCount = 2; * * $(document).ready(function() { * $('#example').dataTable(); * }); * * function fnClickAddRow() { * $('#example').dataTable().fnAddData([* giCount+".1", * giCount+".2", * giCount+".3", * giCount+".4"] *); * * giCount++; * }
1020. If not being removed from the document, make all columns visible
1021. In rtl text layout, some browsers (most, but not all) will place the scrollbar on the left, rather than the right.
1022. Language definitions
1023. * * __Deprecated__ The functionality provided by this parameter has now been * superseded by that provided through `ajax`, which should be used instead. * * Set the HTTP method that is used to make the Ajax call for server-side * processing or Ajax sourced data. * @type string * @default GET * * @dtopt Options * @dtopt Server-side * @name DataTable.defaults.serverMethod * * @deprecated 1.10. Please use `ajax` for this functionality now.
1024. Map the initialisation options onto the settings object
1025. Have to add class here as order event isn't called
1026. no search
1027. Single column - already sorting on this column, modify the sort
1028. * * The DataTables object for this table * @type object * @default null
1029. * * Column index. This could be worked out on-the-fly with $.inArray, but it * is faster to just hold it as a variable * @type integer * @default null
1030. * * Details the actions that will be taken when the user types into the * filtering input text box. The variable "_INPUT_", if used in the string, * is replaced with the HTML text box for the filtering input allowing * control over where it appears in the string. If "_INPUT_" is not given * then the input box is appended to the string automatically. * @type string * @default Search: * * @dtopt Language * @name DataTable.defaults.language.search * * @example * // Input text box will be appended at the end automatically * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "search": "Filter records:" * } * }); * }); * * @example * // Specify where the filter should appear * $(document).ready(function() { * $('#example').dataTable({ * "language": { * "search": "Apply filter _INPUT_ to table" * } * }); * });
1031. no unshift in ie6
1032. Row + column selector
1033. * * State load event, fired when the table is loading state from the stored * data, but prior to the settings object being modified by the saved state * - allowing modification of the saved state is required or loading of * state for a plug-in. * @name DataTable#stateLoadParams.dt * @event * @param {event} e jQuery event object * @param {object} oSettings DataTables settings object * @param {object} json The saved state information
1034. Let any modules know about the draw hold position state (used by scrolling internally)
1035. Not interested in doing column width calculation if auto-width is disabled
1036. Cell index objects in first parameter
1037. If a string is given in between the array notation indicators, that is used to join the strings together, otherwise an array is returned
1038. event, handler
1039. * * Allows a default value to be given for a column's data, and will be used * whenever a null data source is encountered (this can be because `data` * is set to null, or because the data source itself is null). * @type string * @default null * * @name DataTable.defaults.column.defaultContent * @dtopt Columns * * @example * // Using `columnDefs` * $(document).ready(function() { * $('#example').dataTable({ * "columnDefs": [* { * "data": null, * "defaultContent": "Edit", * "targets":

[-1] * } *] * }); * }); * * @example * // Using `columns` * $(document).ready(function() { * $('#example').dataTable({ * "columns": [* null, * null, * null, * { * "data": null, * "defaultContent": "Edit" * } *] * }); * });
1040. otherwise a 2D array was passed in
1041. * * Denote if the original data source was from the DOM, or the data source * object. This is used for invalidating data, so DataTables can * automatically read data from the original source, unless uninstructed * otherwise. * @type string * @default null * @private
1042. Plain numbers
1043. For each column, spin over the
1044. We've been asked to save data to an array, but it isn't array data to be saved. Best that can be done is to just save the value.
1045. Save the filtering values
1046. DataTables 1.9- compatibility
1047. Column visibility change - update the colspan
1048. * * Load data from the newly set Ajax URL. Note that this method is only * available when `ajax.url()` is used to set a URL. Additionally, this method * has the same effect as calling `ajax.reload()` but is provided for * convenience when setting a new URL. Like `ajax.reload()` it will * automatically redraw the table once the remote data has been loaded. * * @returns {DataTables.Api} this
1049. show
1050. Load the data needed for the sort, for each cell
1051. * * Attach a sort listener to an element for a given column * * @param {node|jQuery|string} node Identifier for the element(s) to attach the * listener to. This can take the form of a single DOM node, a jQuery * collection of nodes or a jQuery selector which will identify the node(s). * @param {integer} column the column that a click on this node will sort on * @param {function} [callback] callback function when sort is run * @returns {DataTables.Api} this
1052. * * Viewport height for vertical scrolling. Vertical scrolling is disabled * if an empty string. * Note that this parameter will be set by the initialisation routine. To * set a default use {@link DataTable.defaults}. * @type string
1053. * * Array of callback functions for state loading. Each array element is an * object with the following parameters: * <ul> * <li>function:fn - function to call. Takes two parameters, oSettings * and the object stored. May return false to cancel state loading</li> * <li>string:sName - name of callback</li> * </ul> * @type array * @default []
1054. Otherwise create a row with a wrapper
1055. If the were originally stripe classes - then we add them back here. Note this is not fool proof (for example if not all rows had stripe classes - but it's a good effort without getting carried away
1056. Display start point, taking into account the save saving
1057. original header is in its own table
1058. No additional mark-up required Attach a sort listener to update on sort - note that using the `DT` namespace will allow the event to be removed automatically on destroy, while the `dt` namespaced event is the one we are listening for
1059. Compatibility for browsers without EMCA-252-5 (JS 1.6)
1060. * * Filter the table using both the global filter and column based filtering * @param {object} oSettings dataTables settings object * @param {object} oSearch search information * @param {int} [iForce] force a research of the master array (1) or not (undefined or 0) * @memberof DataTable#oApi
1061. use a new object, in case someone changes the values
1062. On each draw, insert the required elements into the document
1063. * * This parameter has been replaced by `data` in DataTables to ensure naming * consistency. `dataProp` can still be used, as there is backwards * compatibility in DataTables for this option, but it is strongly * recommended that you use `data` in preference to `dataProp`. * @name DataTable.defaults.column.dataProp
1064. On redraw - align columns
1065. globals $,require,jQuery,define,_selector_run,_selector_opts,_selector_first,_selector_row_indexes,_ext,_Api,_api_register,_api_registerPlural,_re_new_lines,_re_html,_re_
1066. Loop over the definitions array - loop in reverse so first instance has priority
1067. Escape regular expression special characters
1068. not an expensive call
1069. All cells and function selectors
1070. Add columns that we don't yet know about
1071. Take the brutal approach to cancelling text selection
1072. Find the last white space character in the string
1073. Order, search and type get the original data
1074. * * This data rendering helper method can be useful for cases where you have * potentially large data strings to be shown in a column that is restricted by * width. The data for the column is still fully searchable and sortable, but if * it is longer than a give number of characters, it will be truncated and * shown with ellipsis. A browser provided tooltip will show the full string * to the end user on mouse hover of the cell. * * This function should be used with the `dt-init columns.render` configuration * option of DataTables. * * It accepts three parameters: * * 1. `-type integer` - The number of characters to restrict the displayed data * to. * 2. `-type boolean` (optional - default `false`) - Indicate if the truncation * of the string should not occur in the middle of a word (`true`) or if it * can (`false`). This can allow the display of strings to look nicer, at the * expense of showing less characters. * 2. `-type boolean` (optional - default `false`) - Escape HTML entities * (`true`) or not (`false` - default). * * @name ellipsis * @summary Restrict output data to a particular length, showing anything * longer with ellipsis and a browser provided tooltip on hover. * @author [Allan Jardine](http://datatables.net) * @requires DataTables 1.10+ * * @returns {Number} Calculated average * * @example * // Restrict a column to 17 characters, don't split words * $('#example').DataTable({ * columnDefs: [{ * targets: 1, * render: $.fn.dataTable.render.ellipsis(17, true) * }] * }); * * @example * // Restrict a column to 10 characters, do split words * $('#example').DataTable({ * columnDefs: [{ * targets: 2, * render: $.fn.dataTable.render.ellipsis(10) * }] * });
1075. Protect against uncontrolled HTML input
1076. cast numbers
1077. * * Return zero for fractions. Partial numbers don't make sense in metrics.
1078. running compactions with queued in parenthesis
1079. running scans with queued in parenthesis
1080. combo string with running value and number queued in parenthesis
1081. * * Return zero for fractions. Partial numbers don't make sense in metrics.
1082. If the number is smaller than the base, return the number with no suffix
1083. * * Formats the timestamp nicely
1084. * * Formats the log level as HTML
1085. Divides the number by the equivalent suffix number
1086. if the number is a fraction keep to 2 decimal places
1087. Create a table for tserver list

git_commits:

1. **summary:** ACCUMULO-4771 Use DataTables in Monitor (#352)
   message: ACCUMULO-4771 Use DataTables in Monitor (#352) * Changed larger tables to use DataTables javascript library * Removed custom javascript sorting for consistency * Cleaned up page titles and removed redundant subtitles

github_issues:

github_issues_comments:

github_pulls:

1. **title:** ACCUMULO-4771 Implement DataTables in Monitor
   body: * Drastically reduces our custom JS code. * Created a new rest endpoint (/dataTables) that could replace a lot of the code in TablesResource * Made improvements to the TableInformation JSON object * Re-purposed TableInformationList for use in DataTables (was only used in XmlInformation) * select2 is

eliminated but currently no way to search on default (empty) namespace. If we still want this I believe this can be done using type based search https://datatables.net/plug-ins/filtering/ but I haven't played around with it yet. I only implemented DataTables for the "tables" page so I'm soliciting feedback before making changes to all tables on the Monitor

github_pulls_comments:

1. What do you guys think of 5373bb25fb49a83fd8fa1f565b683fe62738110f ? It is a wrapper for DataTables so any JSON object we want to render we could just wrap with the class.
2. The more I think about wrapping the current REST endpoints, the more I realize that just complicates the backend. I am thinking the Monitor should be as simple as possible with everything useful that was in the old one.
3. I am going to close this PR so I can rebase, dropping the unnecessary commits.

github_pulls_reviews:

1. Could be simplified to: ```java return dirtyNumber < 1 ? 0 : dirtyNumber; ```
2. Thanks. I had originally thought this method was gonna do more but the JS functions took care of the rest.
3. It might be good to deduplicate this REST endpoint with any others that might be redundant.
4. Not sure this comment will make sense in the future, because somebody editing this might not understand what is meant by "populate DataTables". Also, it's probably not needed to explain prescriptively how an API is used, as that can change over time. It's better to focus comments on what it provides, rather than what calls it.
5. Is this vanilla, or did you have to tweak it? Things in the external folder should correspond to an upstream, so I'm just checking. If we have to tweak it, we can tweak it in our own CSS file or inline.
6. I agree. This could replace TablesResource but I kept it since it had the namespace stuff in it as well.
7. Good point. I put this comment to explain the changes in the class below but perhaps I should just comment next to the "data" field as to why its called something as generic as data. I tried to get DataTables to work with the JSON object as "table" but couldn't and it wasn't pretty.
8. It is vanilla, downloaded as-is from the site.
9. FYI I figured out how to tell DataTables to look for "table" vs "data". Its easy I was just using the wrong json parameter. This will make it so less changes are needed and we can use the POJOs already returned by our rest endpoints.

jira_issues:

jira_issues_comments: