

# The Programming Junkie

## SDL 2.0 API Quick Reference

by Dan Bechard | March 22, 2023

### Overview

Below is a cheatsheet / quick reference for the SDL 2 API. I made this cheatsheet to fill a void that I felt existed in SDL's documentation compared to [Raylib](#).

Please refer to the [official SDL documentation](#) for the most up-to-date API information:

sdl2\_quickref.c

#### CONTEN

- Overview
- sdl2\_quickr

#### Code

```
1 // NOTE: For best viewing experience, click "Raw" in the top right of the Gist page, or download
2 // this file and view it in your favorite text editor with syntax highlighting!
3
4 //=====
5 /// Title      : SDL 2.0 API Quick Reference
6 /// Author     : https://github.com/dbechrd/
7 /// Last updated : Jan 8, 2023
8 //=====
9
10 // Based on: https://wiki.libsdl.org/SDL2/APIByCategory (retrieved Jan 8, 2023)
11 // Full source: https://github.com/libsdl-org/SDL/tree/SDL2 (SDL2 branch)
12 // ASCII art generated by: https://patorjk.com/software/taag/#p=display&f=ANSI%20Shadow&t=Shared%20Object%20Support (with modified 'S' for read
13
14
15 //
16 //
17 //
18 //
19 //
20 //
21
22 //-----
23 /// Init & Quit (SDL.h)
24 //-----
25 int      SDL_Init      (Uint32 flags); // Initialize the SDL library.
26 int      SDL_InitSubSystem (Uint32 flags); // Compatibility function to initialize the SDL library.
27 void     SDL_QuitSubSystem (Uint32 flags); // Shut down specific SDL subsystems.
28 Uint32   SDL_WasInit    (Uint32 flags); // Get a mask of the specified subsystems which are currently initialized.
29 void     SDL_Quit      (void); // Clean up all initialized subsystems.
30
31 //-----
32 /// Config Variables (SDL_hints.h)
33 //-----
34 SDL_bool  SDL_SetHintWithPriority (const char *name, const char *value, SDL_HintPriority priority); // Set a hint with a specific priority.
35 SDL_bool  SDL_SetHint            (const char *name, const char *value); // Set a hint with normal priority.
36 SDL_bool  SDL_ResetHint          (const char *name); // Reset a hint to the default value.
37 void      SDL_ResetHints         (void); // Reset all hints to the default value.
38 const char * SDL_GetHint         (const char *name); // Get the value of a hint.
39 SDL_bool  SDL_GetHintBoolean     (const char *name, SDL_bool default_value); // Get the boolean value of a hint value.
40 void      SDL_AddHintCallback    (const char *name, SDL_HintCallback callback, void *userdata); // Add a function to watch a particular hint.
41 void      SDL_DelHintCallback    (const char *name, SDL_HintCallback callback, void *userdata); // Remove a function watching a particular hint.
42 void      SDL_ClearHints        (void); // Clear all hints.
43
44 //-----
45 /// Error Handling (SDL_error.h)
46 //-----
47 int      SDL_SetError      (const char *fmt, ...); // Set the SDL error message for the current thread.
48 const char * SDL_GetError  (void); // Retrieve a message about the last error that occurred on the current thread.
49 char *     SDL_GetErrorMsg (char *errstr, int maxlen); // Get the last error message that was set for the current thread.
50 void      SDL_ClearError   (void); // Clear any previous error message for this thread.
51
52 //-----
53 /// Logging (SDL_log.h)
54 //-----
55 void      SDL_LogSetAllPriority (SDL_LogPriority priority); // Set the priority of all logging.
56 void      SDL_LogSetPriority   (int category, SDL_LogPriority priority); // Set the priority of a particular category.
57 SDL_LogPriority SDL_LogGetPriority (int category); // Get the priority of a particular category.
58 void      SDL_LogResetPriorities (void); // Reset all priorities to the default value.
59 void      SDL_Log             (const char *fmt, ...); // Log a message with SDL_LOG_DEFAULT.
60 void      SDL_LogVerbose      (int category, const char *fmt, ...); // Log a message with SDL_LOG_VERBOSE.
61 void      SDL_LogDebug        (int category, const char *fmt, ...); // Log a message with SDL_LOG_DEBUG.
62 void      SDL_LogInfo         (int category, const char *fmt, ...); // Log a message with SDL_LOG_INFO.
63 void      SDL_LogWarn         (int category, const char *fmt, ...); // Log a message with SDL_LOG_WARN.
64 void      SDL_LogError        (int category, const char *fmt, ...); // Log a message with SDL_LOG_ERROR.
65 void      SDL_LogCritical     (int category, const char *fmt, ...); // Log a message with SDL_LOG_CRITICAL.
66 void      SDL_LogMessage      (int category, SDL_LogPriority priority, const char *fmt, ...); // Log a message with the specified category and priority.
67 void      SDL_LogMessageV     (int category, SDL_LogPriority priority, const char *fmt, va_list ap); // Log a message with the specified category and priority.
68 void      SDL_LogOutputFunction (SDL_LogOutputFunction callback, void **userdata); // Get the current log output function.
69 void      SDL_LogSetOutputFunction (SDL_LogOutputFunction callback, void *userdata); // Replace the default log output function.
70
71 //-----
72 /// Assertions (SDL_assert.h)
73 //-----
74 void      SDL_SetAssertionHandler (SDL_AssertionHandler handler, void *userdata); // Set an application-defined assertion handler.
75 SDL_AssertionHandler SDL_GetDefaultAssertionHandler (void); // Get the default assertion handler.
76 SDL_AssertionHandler SDL_GetAssertionHandler (void **userdata); // Get the current assertion handler.
77 const SDL_AssertData * SDL_GetAssertionReport (void); // Get a list of all assertion failures.
78 void      SDL_ResetAssertionReport (void); // Clear the list of all assertion failures.
79
80 //-----
81 /// SDL Version (SDL_version.h, SDL_revision.h)
82 //-----
83 void      SDL_GetVersion      (SDL_version *ver); // Get the version of SDL that is linked against your program.
84 const char * SDL_GetRevision  (void); // Get the code revision of SDL that is linked against your program.
85
86
87 //
88 //
89 //
90 //
91 //
92 //
93
94 //-----
95 /// Displays and Windows (SDL_video.h)
96 //-----
97 int      SDL_GetNumVideoDrivers (void); // Get the number of video drivers.
98 const char * SDL_GetVideoDriver (int index); // Get the name of the video driver.
99 int      SDL_VideoInit          (const char *driver_name); // Initialize the video subsystem.
100 void     SDL_VideoQuit          (void); // Shut down the video subsystem.
101 const char * SDL_GetCurrentVideoDriver (void); // Get the name of the current video driver.
```

```
102 int SDL_GetNumVideoDisplays (void); // Get
103 const char * SDL_GetDisplayName (int displayIndex); // Get
104 int SDL_GetDisplayBounds (int displayIndex, SDL_Rect *rect); // Get
105 int SDL_GetDisplayUsableBounds (int displayIndex, SDL_Rect *rect); // Get
106 int SDL_GetDisplayDPI (int displayIndex, float *ddpi, float *hdp, float *vdpi); // Get
107 SDL_DisplayOrientation SDL_GetDisplayOrientation (int displayIndex); // Get
108 int SDL_GetNumDisplayModes (int displayIndex); // Get
109 int SDL_GetDisplayMode (int displayIndex, int modeIndex, SDL_DisplayMode *mode); // Get
110 int SDL_GetDesktopDisplayMode (int displayIndex, SDL_DisplayMode *mode); // Get
111 int SDL_GetCurrentDisplayMode (int displayIndex, SDL_DisplayMode *mode); // Get
112 SDL_DisplayMode * SDL_GetClosestDisplayMode (int displayIndex, const SDL_DisplayMode *mode, SDL_DisplayMode *closest); // Get
113 int SDL_GetPointDisplayIndex (const SDL_Point *point); // Get
114 int SDL_GetRectDisplayIndex (const SDL_Rect *rect); // Get
115 int SDL_GetWindowDisplayIndex (SDL_Window *window); // Get
116 int SDL_SetWindowDisplayMode (SDL_Window *window, const SDL_DisplayMode *mode); // Set
117 int SDL_GetWindowDisplayMode (SDL_Window *window, SDL_DisplayMode *mode); // Que
118 void* SDL_GetWindowICCProfile (SDL_Window *window, size_t *size); // Get
119 Uint32 SDL_GetWindowPixelFormat (SDL_Window *window); // Get
120 SDL_Window * SDL_CreateWindow (const char *title, int x, int y, int w, int h, Uint32 flags); // Cra
121 SDL_Window * SDL_CreateWindowFrom (const void *data); // Cra
122 Uint32 SDL_GetWindowID (SDL_Window *window); // Get
123 SDL_Window * SDL_GetWindowFromID (Uint32 id); // Get
124 Uint32 SDL_GetWindowFlags (SDL_Window *window); // Get
125 void SDL_SetWindowTitle (SDL_Window *window, const char *title); // Set
126 const char * SDL_GetWindowTitle (SDL_Window *window); // Get
127 void SDL_SetWindowIcon (SDL_Window *window, SDL_Surface *icon); // Set
128 void* SDL_SetWindowData (SDL_Window *window, const char *name, void *userdata); // Ass
129 void * SDL_GetWindowData (SDL_Window *window, const char *name); // Ret
130 void SDL_SetWindowPosition (SDL_Window *window, int x, int y); // Set
131 void SDL_GetWindowPosition (SDL_Window *window, int *x, int *y); // Get
132 void SDL_SetWindowSize (SDL_Window *window, int w, int h); // Set
133 void SDL_GetWindowSize (SDL_Window *window, int *w, int *h); // Get
134 int SDL_GetWindowBordersSize (SDL_Window *window, int *top, int *left, int *bottom, int *right); // Get
135 void SDL_GetWindowSizeInPixels (SDL_Window *window, int *w, int *h); // Get
136 void SDL_SetWindowMinimumSize (SDL_Window *window, int min_w, int min_h); // Set
137 void SDL_GetWindowMinimumSize (SDL_Window *window, int *w, int *h); // Get
138 void SDL_SetWindowMaximumSize (SDL_Window *window, int max_w, int max_h); // Set
139 void SDL_GetWindowMaximumSize (SDL_Window *window, int *w, int *h); // Get
140 void SDL_SetWindowBordered (SDL_Window *window, SDL_bool bordered); // Set
141 void SDL_SetWindowResizable (SDL_Window *window, SDL_bool resizable); // Set
142 void SDL_SetWindowAlwaysOnTop (SDL_Window *window, SDL_bool on_top); // Set
143 void SDL_ShowWindow (SDL_Window *window); // Sho
144 void SDL_HideWindow (SDL_Window *window); // Hid
145 void SDL_RaiseWindow (SDL_Window *window); // Rai
146 void SDL_MaximizeWindow (SDL_Window *window); // Mak
147 void SDL_MinimizeWindow (SDL_Window *window); // Min
148 void SDL_RestoreWindow (SDL_Window *window); // Res
149 int SDL_SetWindowFullscreen (SDL_Window *window, Uint32 flags); // Set
150 SDL_Surface * SDL_GetWindowSurface (SDL_Window *window); // Get
151 int SDL_UpdateWindowSurface (SDL_Window *window); // Cop
152 int SDL_UpdateWindowSurfaceRects (SDL_Window *window, const SDL_Rect *rects, int numrects); // Cop
153 void SDL_SetWindowGrab (SDL_Window *window, SDL_bool grabbed); // Set
154 void SDL_SetWindowKeyboardGrab (SDL_Window *window, SDL_bool grabbed); // Set
155 void SDL_SetWindowMouseGrab (SDL_Window *window, SDL_bool grabbed); // Set
156 SDL_bool SDL_GetWindowGrab (SDL_Window *window); // Get
157 SDL_bool SDL_GetWindowKeyboardGrab (SDL_Window *window); // Get
158 SDL_bool SDL_GetWindowMouseGrab (SDL_Window *window); // Get
159 SDL_Window * SDL_GetGrabbedWindow (void); // Get
160 int SDL_SetWindowMouseRect (SDL_Window *window, const SDL_Rect *rect); // Con
161 const SDL_Rect * SDL_GetWindowMouseRect (SDL_Window *window); // Get
162 int SDL_SetWindowBrightness (SDL_Window *window, float brightness); // Set
163 float SDL_GetWindowBrightness (SDL_Window *window); // Get
164 int SDL_SetWindowOpacity (SDL_Window *window, float opacity); // Set
165 int SDL_GetWindowOpacity (SDL_Window *window, float *out_opacity); // Get
166 int SDL_SetWindowModalFor (SDL_Window *modal_window, SDL_Window *parent_window); // Set
167 int SDL_SetWindowInputFocus (SDL_Window *window); // Exp
168 int SDL_SetWindowGammaRamp (SDL_Window *window, const Uint16 *red, const Uint16 *green, const Uint16 *blue); // Set
169 int SDL_GetWindowGammaRamp (SDL_Window *window, Uint16 *red, Uint16 *green, Uint16 *blue); // Get
170 int SDL_SetWindowHitTest (SDL_Window *window, SDL_HitTest callback, void *callback_data); // Pro
171 int SDL_FlashWindow (SDL_Window *window, SDL_FlashOperation operation); // Req
172 void SDL_DestroyWindow (SDL_Window *window); // Des
173 SDL_bool SDL_IsScreenSaverEnabled (void); // Che
174 void SDL_EnableScreenSaver (void); // All
175 void SDL_DisableScreenSaver (void); // Pre
176
177 ///-----
178 /// OpenGL support functions (SDL_video.h)
179 ///-----
180 int SDL_GL_LoadLibrary (const char *path); // Dynamically load an OpenGL library.
181 void * SDL_GL_GetProcAddress (const char *proc); // Get an OpenGL function by name.
182 void SDL_GL_UnloadLibrary (void); // Unload the OpenGL library previously loaded by SDL_G
183 SDL_bool SDL_GL_ExtensionSupported (const char *extension); // Check if an OpenGL extension is supported for the cu
184 void SDL_GL_ResetAttributes (void); // Reset all previously set OpenGL context attributes t
185 int SDL_GL_SetAttribute (SDL_GLattr attr, int value); // Set an OpenGL window attribute before window creatio
186 int SDL_GL_GetAttribute (SDL_GLattr attr, int *value); // Get the actual value for an attribute from the curre
187 SDL_GLContext SDL_GL_CreateContext (SDL_Window *window); // Create an OpenGL context for an OpenGL window, and m
188 int SDL_GL_MakeCurrent (SDL_Window *window, SDL_GLContext context); // Set up an OpenGL context for rendering into an OpenG
189 SDL_Window * SDL_GL_GetCurrentWindow (void); // Get the currently active OpenGL window.
190 SDL_GLContext SDL_GL_GetCurrentContext (void); // Get the currently active OpenGL context.
191 void SDL_GL_GetDrawableSize (SDL_Window *window, int *w, int *h); // Get the size of a window's underlying drawable in pi
192 int SDL_GL_SetSwapInterval (int interval); // Set the swap interval for the current OpenGL context
193 int SDL_GL_GetSwapInterval (void); // Get the swap interval for the current OpenGL context
194 void SDL_GL_SwapWindow (SDL_Window *window); // Update a window with OpenGL rendering.
195 void SDL_GL_DeleteContext (SDL_GLContext context); // Delete an OpenGL context.
196
197 ///-----
198 /// 2D Accelerated Rendering (SDL_render.h)
199 ///-----
200 int SDL_GetNumRenderDrivers (void); // Ge
201 int SDL_GetRenderDriverInfo (int index, SDL_RendererInfo *info); // Ge
202 int SDL_CreateWindowAndRenderer (int width, int height, Uint32 window_flags, SDL_Window **window,
203 SDL_Renderer **renderer); // Cr
204 SDL_Renderer * SDL_CreateRenderer (SDL_Window *window, int index, Uint32 flags); // Cr
205 SDL_Renderer * SDL_CreateSoftwareRenderer (SDL_Surface *surface); // Cr
206 SDL_Renderer * SDL_GetRenderer (SDL_Window *window); // Ge
207 SDL_Window * SDL_RendererGetWindow (SDL_Renderer *renderer); // Ge
208 int SDL_GetRendererInfo (SDL_Renderer *renderer, SDL_RendererInfo *info); // Ge
209 int SDL_GetRendererOutputSize (SDL_Renderer *renderer, int *w, int *h); // Ge
210 SDL_Texture * SDL_CreateTexture (SDL_Renderer *renderer, Uint32 format, int access, int w, int h); // Cr
211 SDL_Texture * SDL_CreateTextureFromSurface (SDL_Renderer *renderer, SDL_Surface *surface); // Cr
212 int SDL_QueryTexture (SDL_Texture *texture, Uint32 *format, int *access, int *w, int *h); // Qu
213 int SDL_SetTextureColorMod (SDL_Texture *texture, Uint8 r, Uint8 g, Uint8 b); // Se
214 int SDL_GetTextureColorMod (SDL_Texture *texture, Uint8 *r, Uint8 *g, Uint8 *b); // Ge
215 int SDL_SetTextureAlphaMod (SDL_Texture *texture, Uint8 alpha); // Se
216 int SDL_GetTextureAlphaMod (SDL_Texture *texture, Uint8 *alpha); // Ge
217 int SDL_SetTextureBlendMode (SDL_Texture *texture, SDL_BlendMode blendMode); // Se
218 int SDL_GetTextureBlendMode (SDL_Texture *texture, SDL_BlendMode *blendMode); // Ge
219 int SDL_SetTextureScaleMode (SDL_Texture *texture, SDL_ScaleMode scaleMode); // Se
220 int SDL_GetTextureScaleMode (SDL_Texture *texture, SDL_ScaleMode *scaleMode); // Ge
221 int SDL_SetTextureUserData (SDL_Texture *texture, void *userdata); // As
222 void * SDL_GetTextureUserData (SDL_Texture *texture); // Ge
223 int SDL_UpdateTexture (SDL_Texture *texture, const SDL_Rect *rect, const void *pixels, int pitch); // Up
224 int SDL_UpdateYUVTexture (SDL_Texture *texture, const SDL_Rect *rect, const Uint8 *Yplane, int Ypitch,
225 const Uint8 *Uplane, int Upitch, const Uint8 *Vplane, int Vpitch); // Up
226 int SDL_UpdateNVTexture (SDL_Texture *texture, const SDL_Rect *rect, const Uint8 *Yplane, int Ypitch,
227 const Uint8 *UVplane, int UVpitch); // Up
228 int SDL_LockTexture (SDL_Texture *texture, const SDL_Rect *rect, void **pixels, int *pitch); // Lo
```

```
229 int SDL_LockTextureToSurface (SDL_Texture *texture, const SDL_Rect *rect, SDL_Surface **surface); // Lo
230 void SDL_UnlockTexture (SDL_Texture *texture); // Un
231 SDL_bool SDL_RenderTargetSupported (SDL_Renderer *renderer); // De
232 int SDL_SetRenderTarget (SDL_Renderer *renderer, SDL_Texture *texture); // Se
233 SDL_Texture * SDL_GetRenderTarget (SDL_Renderer *renderer); // Ge
234 int SDL_RenderSetLogicalSize (SDL_Renderer *renderer, int w, int h); // Se
235 void SDL_RenderGetLogicalSize (SDL_Renderer *renderer, int *w, int *h); // Ge
236 int SDL_RenderSetIntegerScale (SDL_Renderer *renderer, SDL_bool enable); // Se
237 SDL_bool SDL_RenderGetIntegerScale (SDL_Renderer *renderer); // Ge
238 int SDL_RenderSetViewport (SDL_Renderer *renderer, const SDL_Rect *rect); // Se
239 void SDL_RenderGetViewport (SDL_Renderer *renderer, SDL_Rect *rect); // Ge
240 int SDL_RenderSetClipRect (SDL_Renderer *renderer, const SDL_Rect *rect); // Se
241 void SDL_RenderGetClipRect (SDL_Renderer *renderer, SDL_Rect *rect); // Ge
242 SDL_bool SDL_RenderIsClipEnabled (SDL_Renderer *renderer); // Ge
243 int SDL_RenderSetScale (SDL_Renderer *renderer, float scaleX, float scaleY); // Se
244 void SDL_RenderGetScale (SDL_Renderer *renderer, float *scaleX, float *scaleY); // Ge
245 void SDL_RenderWindowToLogical (SDL_Renderer *renderer, int windowX, int windowY, float *logicalX, float *logicalY); // Ge
246 void SDL_RenderLogicalToWindow (SDL_Renderer *renderer, float logicalX, float logicalY, int *windowX, int *windowY); // Ge
247 int SDL_SetRenderDrawColor (SDL_Renderer *renderer, Uint8 r, Uint8 g, Uint8 b, Uint8 a); // Se
248 int SDL_GetRenderDrawColor (SDL_Renderer *renderer, Uint8 *r, Uint8 *g, Uint8 *b, Uint8 *a); // Ge
249 int SDL_SetRenderDrawBlendMode (SDL_Renderer *renderer, SDL_BlendMode blendMode); // Se
250 int SDL_GetRenderDrawBlendMode (SDL_Renderer *renderer, SDL_BlendMode *blendMode); // Ge
251 int SDL_RenderClear (SDL_Renderer *renderer); // CL
252 int SDL_RenderDrawPoint (SDL_Renderer *renderer, int x, int y); // Dr
253 int SDL_RenderDrawPoints (SDL_Renderer *renderer, const SDL_Point *points, int count); // Dr
254 int SDL_RenderDrawLine (SDL_Renderer *renderer, int x1, int y1, int x2, int y2); // Dr
255 int SDL_RenderDrawLines (SDL_Renderer *renderer, const SDL_Point *points, int count); // Dr
256 int SDL_RenderDrawRect (SDL_Renderer *renderer, const SDL_Rect *rect); // Dr
257 int SDL_RenderDrawRects (SDL_Renderer *renderer, const SDL_Rect *rects, int count); // Dr
258 int SDL_RenderFillRect (SDL_Renderer *renderer, const SDL_Rect *rect); // Fi
259 int SDL_RenderFillRects (SDL_Renderer *renderer, const SDL_Rect *rects, int count); // Fi
260 int SDL_RenderCopy (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Rect *srcRect, // Co
261 const SDL_Rect *dstRect); // Co
262 int SDL_RenderCopyEx (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Rect *srcRect, // Co
263 const SDL_Rect *dstRect, const double angle, const SDL_Point *center, // Co
264 const SDL_RendererFlip flip); // Co
265 int SDL_RenderDrawPointF (SDL_Renderer *renderer, float x, float y); // Dr
266 int SDL_RenderDrawPointsF (SDL_Renderer *renderer, const SDL_FPoint *points, int count); // Dr
267 int SDL_RenderDrawLineF (SDL_Renderer *renderer, float x1, float y1, float x2, float y2); // Dr
268 int SDL_RenderDrawLinesF (SDL_Renderer *renderer, const SDL_FPoint *points, int count); // Dr
269 int SDL_RenderDrawRectF (SDL_Renderer *renderer, const SDL_FRect *rect); // Dr
270 int SDL_RenderDrawRectsF (SDL_Renderer *renderer, const SDL_FRect *rects, int count); // Dr
271 int SDL_RenderFillRectF (SDL_Renderer *renderer, const SDL_FRect *rect); // Fi
272 int SDL_RenderFillRectsF (SDL_Renderer *renderer, const SDL_FRect *rects, int count); // Fi
273 int SDL_RenderCopyF (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Rect *srcRect, // Co
274 const SDL_FRect *dstRect); // Co
275 int SDL_RenderCopyExF (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Rect *srcRect, // Co
276 const SDL_FRect *dstRect, const double angle, const SDL_FPoint *center, // Co
277 const SDL_RendererFlip flip); // Co
278 int SDL_RenderGeometry (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Vertex *vertices, // Re
279 int num_vertices, const int *indices, int num_indices); // Re
280 int SDL_RenderGeometryRaw (SDL_Renderer *renderer, SDL_Texture *texture, const float *xy, int xy_stride, // Re
281 const SDL_Color *color, int color_stride, const float *uv, int uv_stride, // Re
282 int num_vertices, const void *indices, int num_indices, int size_indices); // Re
283 int SDL_RenderReadPixels (SDL_Renderer *renderer, const SDL_Rect *rect, Uint32 format, void *pixels, int pitch); // Re
284 void SDL_RenderPresent (SDL_Renderer *renderer); // Up
285 void SDL_DestroyTexture (SDL_Texture *texture); // De
286 void SDL_DestroyRenderer (SDL_Renderer *renderer); // De
287 int SDL_RenderFlush (SDL_Renderer *renderer); // Fl
288 int SDL_GL_BindTexture (SDL_Texture *texture, float *texw, float *texh); // B3
289 int SDL_GL_UnbindTexture (SDL_Texture *texture); // Un
290 void * SDL_RenderGetMetalLayer (SDL_Renderer *renderer); // Ge
291 void * SDL_RenderGetMetalCommandEncoder (SDL_Renderer *renderer); // Ge
292 int SDL_RenderSetVSync (SDL_Renderer *renderer, int vsync); // To
293
294
295 //-----
296 // Pixel Formats and Conversion Routines (SDL_pixels.h)
297 //-----
298 const char* SDL_GetPixelFormatName (Uint32 format); // Get t
299 SDL_bool SDL_PixelFormatEnumToMasks (Uint32 format, int *bpp, Uint32 *Rmask, Uint32 *Gmask, Uint32 *Bmask, Uint32 *Amask); // Conve
300 Uint32 SDL_MasksToPixelFormatEnum (int bpp, Uint32 Rmask, Uint32 Gmask, Uint32 Bmask, Uint32 Amask); // Conve
301 SDL_PixelFormat * SDL_AllocFormat (Uint32 pixel_format); // Creat
302 void SDL_FreeFormat (SDL_PixelFormat *format); // Free
303 SDL_Palette * SDL_AllocPalette (int ncolors); // Creat
304 int SDL_SetPixelFormatPalette (SDL_PixelFormat *format, SDL_Palette *palette); // Set t
305 int SDL_SetPaletteColors (SDL_Palette *palette, const SDL_Color *colors, int firstcolor, int ncolors); // Set a
306 void SDL_FreePalette (SDL_Palette *palette); // Free
307 Uint32 SDL_MapRGB (const SDL_PixelFormat *format, Uint8 r, Uint8 g, Uint8 b); // Map a
308 Uint32 SDL_MapRGBA (const SDL_PixelFormat *format, Uint8 r, Uint8 g, Uint8 b, Uint8 a); // Map a
309 void SDL_GetRGB (Uint32 pixel, const SDL_PixelFormat *format, Uint8 *r, Uint8 *g, Uint8 *b); // Get R
310 void SDL_GetRGBA (Uint32 pixel, const SDL_PixelFormat *format, Uint8 *r, Uint8 *g, Uint8 *b, Uint8 *a); // Get R
311 void SDL_CalculateGammaRamp (float gamma, Uint16 *ramp); // Calcu
312
313 //-----
314 // Rectangle Functions (SDL_rect.h)
315 //-----
316 SDL_bool SDL_PointInRect (const SDL_Point *p, const SDL_Rect *r) // Returns true if point res
317 SDL_bool SDL_RectEmpty (const SDL_Rect *r) // Returns true if the recta
318 SDL_bool SDL_RectEquals (const SDL_Rect *a, const SDL_Rect *b) // Returns true if the two r
319 SDL_bool SDL_HasIntersection (const SDL_Rect *A, const SDL_Rect *B); // Determine whether two rec
320 SDL_bool SDL_IntersectionRect (const SDL_Rect *A, const SDL_Rect *B, SDL_Rect *result); // Calculate the intersectio
321 void SDL_UnionRect (const SDL_Rect *A, const SDL_Rect *B, SDL_Rect *result); // Calculate the union of tw
322 SDL_bool SDL_EnclosePoints (const SDL_Point *points, int count, const SDL_Rect *clip, SDL_Rect *result); // Calculate a minimal recta
323 SDL_bool SDL_IntersectRectAndLine (const SDL_Rect *rect, int *X1, int *Y1, int *X2, int *Y2); // Calculate the intersectio
324
325 //-----
326 // Sub-pixel Rectangle Functions (SDL_rect.h)
327 //-----
328 SDL_bool SDL_PointInFRect (const SDL_FPoint *p, const SDL_FRect *r) // Returns true if point
329 SDL_bool SDL_FRectEmpty (const SDL_FRect *r) // Returns true if the r
330 SDL_bool SDL_FRectEqualsEpsilon (const SDL_FRect *a, const SDL_FRect *b, const float epsilon) // Returns true if the t
331 SDL_bool SDL_FRectEquals (const SDL_FRect *a, const SDL_FRect *b) // Returns true if the t
332 SDL_bool SDL_HasIntersectionF (const SDL_FRect *A, const SDL_FRect *B); // Determine whether two
333 SDL_bool SDL_IntersectionFRect (const SDL_FRect *A, const SDL_FRect *B, SDL_FRect *result); // Calculate the interse
334 void SDL_UnionFRect (const SDL_FRect *A, const SDL_FRect *B, SDL_FRect *result); // Calculate the union o
335 SDL_bool SDL_EncloseFPoints (const SDL_FPoint *points, int count, const SDL_FRect *clip, SDL_FRect *result); // Calculate a minimal r
336 SDL_bool SDL_IntersectFRectAndLine (const SDL_FRect *rect, float *X1, float *Y1, float *X2, float *Y2); // Calculate the interse
337
338 //-----
339 // Surface Creation and Simple Drawing (SDL_surface.h)
340 //-----
341 SDL_Surface * SDL_CreateRGBSurface (Uint32 flags, int width, int height, int depth, Uint32 Rmask, Uint32 Gmask, //
342 Uint32 Bmask, Uint32 Amask); //
343 SDL_Surface * SDL_CreateRGBSurfaceWithFormat (Uint32 flags, int width, int height, int depth, Uint32 format); //
344 SDL_Surface * SDL_CreateRGBSurfaceFrom (void *pixels, int width, int height, int depth, int pitch, Uint32 Rmask, Uint32 //
345 Uint32 Bmask, Uint32 Amask); //
346 SDL_Surface * SDL_CreateRGBSurfaceWithFormatFrom (void *pixels, int width, int height, int depth, int pitch, Uint32 format); //
347 void SDL_FreeSurface (SDL_Surface *surface); //
348 int SDL_SetSurfacePalette (SDL_Surface *surface, SDL_Palette *palette); //
349 int SDL_LockSurface (SDL_Surface *surface); //
350 void SDL_UnlockSurface (SDL_Surface *surface); //
351 SDL_Surface * SDL_LoadBMP_RW (SDL_RWops *src, int freesrc); //
352 int SDL_SaveBMP_RW (SDL_Surface *surface, SDL_RWops *dst, int freedst); //
353 int SDL_SetSurfaceRLE (SDL_Surface *surface, int flag); //
354 SDL_bool SDL_HasSurfaceRLE (SDL_Surface *surface); //
355 int SDL_SetColorKey (SDL_Surface *surface, int flag, Uint32 key); //
```

```
356 SDL_bool SDL_HasColorKey (SDL_Surface *surface);
357 int SDL_GetColorKey (SDL_Surface *surface, Uint32 *key);
358 int SDL_SetSurfaceColorMod (SDL_Surface *surface, Uint8 r, Uint8 g, Uint8 b);
359 int SDL_GetSurfaceColorMod (SDL_Surface *surface, Uint8 *r, Uint8 *g, Uint8 *b);
360 int SDL_SetSurfaceAlphaMod (SDL_Surface *surface, Uint8 alpha);
361 int SDL_GetSurfaceAlphaMod (SDL_Surface *surface, Uint8 *alpha);
362 int SDL_SetSurfaceBlendMode (SDL_Surface *surface, SDL_BlendMode blendMode);
363 int SDL_GetSurfaceBlendMode (SDL_Surface *surface, SDL_BlendMode *blendMode);
364 SDL_bool SDL_SetClipRect (SDL_Surface *surface, const SDL_Rect *rect);
365 void SDL_GetClipRect (SDL_Surface *surface, SDL_Rect *rect);
366 SDL_Surface * SDL_DuplicateSurface (SDL_Surface *surface);
367 SDL_Surface * SDL_ConvertSurface (SDL_Surface *src, const SDL_PixelFormat *fmt, Uint32 flags);
368 SDL_Surface * SDL_ConvertSurfaceFormat (SDL_Surface *src, Uint32 pixel_format, Uint32 flags);
369 int (int width, int height, Uint32 src_format, const void *src, int src_pitch,
370 Uint32 dst_format, void *dst, int dst_pitch);
371 int (int width, int height, Uint32 src_format, const void *src, int src_pitch,
372 Uint32 dst_format, void *dst, int dst_pitch);
373 int SDL_FillRect (SDL_Surface *dst, const SDL_Rect *rect, Uint32 color);
374 int SDL_FillRects (SDL_Surface *dst, const SDL_Rect *rects, int count, Uint32 color);
375 int SDL_UpperBlit (SDL_Surface *src, const SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
376 int SDL_LowerBlit (SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
377 int SDL_SoftStretch (SDL_Surface *src, const SDL_Rect *srcrect, SDL_Surface *dst, const SDL_Rect *dstrect);
378 int SDL_SoftStretchLinear (SDL_Surface *src, const SDL_Rect *srcrect, SDL_Surface *dst, const SDL_Rect *dstrect);
379 int SDL_UpperBlitScaled (SDL_Surface *src, const SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
380 int SDL_LowerBlitScaled (SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
381 void SDL_SetYUVConversionMode (SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
382 SDL_YUV_CONVERSION_MODE SDL_GetYUVConversionMode (SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
383 SDL_YUV_CONVERSION_MODE SDL_GetYUVConversionModeForResolution (SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
384
385 //-----
386 // Platform-specific Window Management (SDL_syswm.h)
387 //-----
388 SDL_bool SDL_GetWindowWMInfo (SDL_Window *window, SDL_SysWMInfo *info); // Get driver-specific information about a window.
389
390 //-----
391 // Clipboard Handling (SDL_clipboard.h)
392 //-----
393 int SDL_SetClipboardText (const char *text); // Put UTF-8 text into the clipboard.
394 char * SDL_GetClipboardText (void); // Get UTF-8 text from the clipboard, which must be freed with SDL_free().
395 SDL_bool SDL_HasClipboardText (void); // Query whether the clipboard exists and contains a non-empty text string.
396 int SDL_SetPrimarySelectionText (const char *text); // Put UTF-8 text into the primary selection.
397 char * SDL_GetPrimarySelectionText (void); // Get UTF-8 text from the primary selection, which must be freed with SDL_free().
398 SDL_bool SDL_HasPrimarySelectionText (void); // Query whether the primary selection exists and contains a non-empty text string.
399
400 //-----
401 // Vulkan Support (SDL_vulkan.h)
402 //-----
403 int SDL_Vulkan_LoadLibrary (const char *path); // Dynamically load the Vulkan library.
404 void * SDL_Vulkan_GetVkGetInstanceProcAddr (void); // Get the address of the 'vkGetInstanceProcAddr' symbol.
405 void SDL_Vulkan_UnloadLibrary (void); // Unload the Vulkan library.
406 SDL_bool SDL_Vulkan_GetInstanceExtensions (SDL_Window *window, unsigned int *pCount, const char **pNames); // Get the names of the Vulkan instance extensions supported by the window's driver.
407 SDL_bool SDL_Vulkan_CreateSurface (SDL_Window *window, VkInstance instance, VkSurfaceKHR *surface); // Create a Vulkan rendering surface.
408 void SDL_Vulkan_GetDrawableSize (SDL_Window *window, int *w, int *h); // Get the size of the window's drawable area.
409
410
411 // EVENTS
412
413
414
415
416
417
418 //-----
419 // Event Handling (SDL_events.h)
420 //-----
421 int SDL_PeepEvents (SDL_Event *events, int numevents, SDL_eventaction action,
422 Uint32 minType, Uint32 maxType); // Check the event queue for messages and optionally return them.
423 SDL_bool SDL_HasEvent (Uint32 type); // Check for the existence of a certain event type in the event queue.
424 SDL_bool SDL_HasEvents (Uint32 minType, Uint32 maxType); // Check for the existence of certain event types in the event queue.
425 void SDL_FlushEvent (Uint32 type); // Clear events of a specific type from the event queue.
426 void SDL_FlushEvents (Uint32 minType, Uint32 maxType); // Clear events of a range of types from the event queue.
427 int SDL_PollEvent (SDL_Event *event); // Poll for currently pending events.
428 int SDL_WaitEvent (SDL_Event *event); // Wait indefinitely for the next available event.
429 int SDL_WaitEventTimeout (SDL_Event *event, int timeout); // Wait until the specified timeout (in milliseconds) for the next event.
430 int SDL_PushEvent (SDL_Event *event); // Add an event to the event queue.
431 void SDL_SetEventFilter (SDL_EventFilter filter, void *userdata); // Set up a filter to process all events before they change internal state.
432 SDL_bool SDL_GetEventFilter (SDL_EventFilter *filter, void **userdata); // Query the current event filter.
433 void SDL_AddEventWatch (SDL_EventFilter filter, void *userdata); // Add a callback to be triggered when an event is added to the event queue.
434 void SDL_DelEventWatch (SDL_EventFilter filter, void *userdata); // Remove an event watch callback added with SDL_AddEventWatch().
435 void SDL_FilterEvents (SDL_EventFilter filter, void *userdata); // Run a specific filter function on the current event queue, removing events that do not pass the filter.
436 Uint8 SDL_EventState (Uint32 type, int state); // Set the state of processing events by type.
437 Uint32 SDL_RegisterEvents (int numevents); // Allocate a set of user-defined events, and return the beginning of the range.
438
439
440 // INPUT
441
442
443
444
445
446
447 //-----
448 // Keyboard Support (SDL_keyboard.h)
449 //-----
450 const Uint8 * SDL_GetKeyboardState (int *numkeys); // Get a snapshot of the current state of the keyboard.
451 SDL_Window * SDL_GetKeyboardFocus (void); // Query the window which currently has keyboard focus.
452 void SDL_ResetKeyboard (void); // Clear the state of the keyboard.
453 SDL_Keysym SDL_GetModState (void); // Get the current key modifier state for the keyboard.
454 void SDL_SetModState (SDL_Keysym modstate); // Set the current key modifier state for the keyboard.
455 SDL_Keysym SDL_GetKeyFromScancode (SDL_Scancode scancode); // Get the key code corresponding to the given scancode according to the current keyboard layout.
456 SDL_Scancode SDL_GetScancodeFromKey (SDL_Keysym key); // Get the scancode corresponding to the given key code according to the current keyboard layout.
457 const char * SDL_GetScancodeName (SDL_Scancode scancode); // Get a human-readable name for a scancode.
458 SDL_Scancode SDL_GetScancodeFromName (const char *name); // Get a scancode from a human-readable name.
459 const char * SDL_GetKeyName (SDL_Keysym key); // Get a human-readable name for a key.
460 SDL_Keysym SDL_GetKeyFromName (const char *name); // Get a key code from a human-readable name.
461 void SDL_StartTextInput (void); // Start accepting Unicode text input events.
462 SDL_bool SDL_IsTextInputActive (void); // Check whether or not Unicode text input events are enabled.
463 void SDL_StopTextInput (void); // Stop receiving any text input events.
464 void SDL_ClearComposition (void); // Dismiss the composition window/IME without disabling the subsystem.
465 SDL_bool SDL_IsTextInputShown (void); // Returns if an IME Composite or Candidate window is currently shown.
466 void SDL_SetTextInputRect (const SDL_Rect *rect); // Set the rectangle used to type Unicode text inputs.
467 SDL_bool SDL_HasScreenKeyboardSupport (void); // Check whether the platform has screen keyboard support.
468 SDL_bool SDL_IsScreenKeyboardShown (SDL_Window *window); // Check whether the screen keyboard is shown for given window.
469
470 //-----
471 // Mouse Support (SDL_mouse.h)
472 //-----
473 SDL_Window * SDL_GetMouseFocus (void); // Get the window which currently has mouse focus.
474 Uint32 SDL_GetMouseState (int *x, int *y); // Retrieve the current state of the mouse.
475 Uint32 SDL_GetGlobalMouseState (int *x, int *y); // Get the current state of the mouse in relation to the global screen.
476 Uint32 SDL_GetRelativeMouseState (int *x, int *y); // Retrieve the relative state of the mouse.
477 void SDL_WarpMouseInWindow (SDL_Window *window, int x, int y); // Move the mouse cursor to the given position within the window.
478 int SDL_WarpMouseGlobal (int x, int y); // Move the mouse to the given position in global screen coordinates.
479 int SDL_SetRelativeMouseMode (SDL_bool enabled); // Set relative mouse mode.
480 int SDL_CaptureMouse (SDL_bool enabled); // Capture the mouse and to track input outside an SDL window.
481 SDL_bool SDL_GetRelativeMouseMode (void); // Query whether relative mouse mode is enabled.
482 SDL_Cursor * SDL_CreateCursor (const Uint8 *data, const Uint8 *mask, int w,
```



```
483         int h, int hot_x, int hot_y); // Create a cursor using the specified bitmap data and
484 SDL_Cursor * SDL_CreateColorCursor(SDL_Surface *surface, int hot_x, int hot_y); // Create a color cursor.
485 SDL_Cursor * SDL_CreateSystemCursor(SDL_SystemCursor id); // Create a system cursor.
486 void SDL_SetCursor(SDL_Cursor *cursor); // Set the active cursor.
487 SDL_Cursor * SDL_GetCursor(void); // Get the active cursor.
488 SDL_Cursor * SDL_GetDefaultCursor(void); // Get the default cursor.
489 void SDL_FreeCursor(SDL_Cursor *cursor); // Free a previously-created cursor.
490 int SDL_ShowCursor(int toggle); // Toggle whether or not the cursor is shown.
491
492 ///-----
493 /// Joystick Support (SDL_joystick.h)
494 ///-----
495 void SDL_LockJoysticks(void); // Locking for multiple threads
496 void SDL_UnlockJoysticks(void); // Unlocking for multiple threads
497 int SDL_NumJoysticks(void); // Count the number of joysticks
498 const char * SDL_JoystickNameForIndex(int device_index); // Get the implementation name
499 const char * SDL_JoystickPathForIndex(int device_index); // Get the implementation path
500 int SDL_JoystickGetDevicePlayerIndex(int device_index); // Get the player index
501 SDL_JoystickGUID SDL_JoystickGetDeviceGUID(int device_index); // Get the implementation GUID
502 Uint16 SDL_JoystickGetDeviceVendor(int device_index); // Get the USB vendor ID
503 Uint16 SDL_JoystickGetDeviceProduct(int device_index); // Get the USB product ID
504 Uint16 SDL_JoystickGetDeviceProductVersion(int device_index); // Get the product version
505 SDL_JoystickType SDL_JoystickGetDeviceType(int device_index); // Get the type of joystick
506 SDL_JoystickID SDL_JoystickGetDeviceInstanceID(int device_index); // Get the instance ID
507 SDL_Joystick * SDL_JoystickOpen(int device_index); // Open a joystick
508 SDL_Joystick * SDL_JoystickFromInstanceID(SDL_JoystickID instance_id); // Get the SDL_Joystick from an instance ID
509 SDL_Joystick * SDL_JoystickFromPlayerIndex(int player_index); // Get the SDL_Joystick from a player index
510 int SDL_JoystickAttachVirtual(SDL_JoystickType type, int naxes, int nbuttons, int nhats); // Attach a new virtual joystick
511 int SDL_JoystickAttachVirtualEx(const SDL_VirtualJoystickDesc *desc); // Attach a new virtual joystick with a description
512 int SDL_JoystickDetachVirtual(int device_index); // Detach a virtual joystick
513 SDL_bool SDL_JoystickIsVirtual(int device_index); // Query whether or not a joystick is virtual
514 int SDL_JoystickSetVirtualAxis(SDL_Joystick *joystick, int axis, Sint16 value); // Set values on an axis
515 int SDL_JoystickSetVirtualButton(SDL_Joystick *joystick, int button, Uint8 value); // Set values on a button
516 int SDL_JoystickSetVirtualHat(SDL_Joystick *joystick, int hat, Uint8 value); // Set values on a hat
517 const char * SDL_JoystickName(SDL_Joystick *joystick); // Get the implementation name
518 const char * SDL_JoystickPath(SDL_Joystick *joystick); // Get the implementation path
519 int SDL_JoystickGetPlayerIndex(SDL_Joystick *joystick); // Get the player index
520 void SDL_JoystickSetPlayerIndex(SDL_Joystick *joystick, int player_index); // Set the player index
521 SDL_JoystickGUID SDL_JoystickGetGUID(SDL_Joystick *joystick); // Get the implementation GUID
522 Uint16 SDL_JoystickGetVendor(SDL_Joystick *joystick); // Get the USB vendor ID
523 Uint16 SDL_JoystickGetProduct(SDL_Joystick *joystick); // Get the USB product ID
524 Uint16 SDL_JoystickGetProductVersion(SDL_Joystick *joystick); // Get the product version
525 Uint16 SDL_JoystickGetFirmwareVersion(SDL_Joystick *joystick); // Get the firmware version
526 const char * SDL_JoystickGetSerial(SDL_Joystick *joystick); // Get the serial number
527 SDL_JoystickType SDL_JoystickGetType(SDL_Joystick *joystick); // Get the type of joystick
528 void SDL_JoystickGetGUIDString(SDL_JoystickGUID guid, char *pszGUID, int cbGUID); // Get an ASCII string representation of a GUID
529 SDL_JoystickGUID SDL_JoystickGetGUIDFromString(const char *pchGUID); // Convert a GUID string to a GUID
530 void SDL_GetJoystickGUIDInfo(SDL_JoystickGUID guid, Uint16 *vendor, Uint16 *product, Uint16 *version, Uint16 *crlc16); // Get the device ID, vendor, product, and version
531 SDL_bool SDL_JoystickGetAttached(SDL_Joystick *joystick); // Get the status of a joystick
532 SDL_JoystickID SDL_JoystickInstanceID(SDL_Joystick *joystick); // Get the instance ID
533 int SDL_JoystickNumAxes(SDL_Joystick *joystick); // Get the number of axes
534 int SDL_JoystickNumBalls(SDL_Joystick *joystick); // Get the number of balls
535 int SDL_JoystickNumHats(SDL_Joystick *joystick); // Get the number of hats
536 int SDL_JoystickNumButtons(SDL_Joystick *joystick); // Get the number of buttons
537 void SDL_JoystickUpdate(void); // Update the current state of the joysticks
538 int SDL_JoystickEventState(int state); // Enable/disable joystick event processing
539 Sint16 SDL_JoystickGetAxis(SDL_Joystick *joystick, int axis); // Get the current axis value
540 SDL_bool SDL_JoystickGetAxisInitialState(SDL_Joystick *joystick, int axis, Sint16 *state); // Get the initial axis value
541 Uint8 SDL_JoystickGetHat(SDL_Joystick *joystick, int hat); // Get the current hat value
542 int SDL_JoystickGetBall(SDL_Joystick *joystick, int ball, int *dx, int *dy); // Get the ball axis coordinates
543 Uint8 SDL_JoystickGetButton(SDL_Joystick *joystick, int button); // Get the current button value
544 int SDL_JoystickRumble(SDL_Joystick *joystick, Uint16 low_frequency_rumble, Uint16 high_frequency_rumble, Uint32 duration_ms); // Start a rumble effect
545 int SDL_JoystickRumbleTriggers(SDL_Joystick *joystick, Uint16 left_rumble, Uint16 right_rumble, Uint32 duration_ms); // Start a rumble effect with triggers
546 SDL_bool SDL_JoystickHasLED(SDL_Joystick *joystick); // Query whether a joystick has an LED
547 SDL_bool SDL_JoystickHasRumble(SDL_Joystick *joystick); // Query whether a joystick has rumble
548 SDL_bool SDL_JoystickHasRumbleTriggers(SDL_Joystick *joystick); // Query whether a joystick has rumble triggers
549 int SDL_JoystickSetLED(SDL_Joystick *joystick, Uint8 red, Uint8 green, Uint8 blue); // Update a joystick LED
550 void SDL_JoystickSendEffect(SDL_Joystick *joystick, const void *data, int size); // Send a joystick effect
551 void SDL_JoystickClose(SDL_Joystick *joystick); // Close a joystick
552 SDL_JoystickPowerLevel SDL_JoystickCurrentPowerLevel(SDL_Joystick *joystick); // Get the battery level
553
554 ///-----
555 /// Game Controller Support (SDL_gamecontroller.h)
556 ///-----
557 int SDL_GameControllerAddMappingsFromRW(SDL_RWops *rw, int freerw); // Add mappings from a file or memory
558 int SDL_GameControllerAddMapping(const char *mappingString); // Add a mapping string
559 int SDL_GameControllerNumMappings(void); // Get the number of mappings
560 char * SDL_GameControllerMappingForIndex(int mapping_index); // Get the implementation name
561 char * SDL_GameControllerMappingForGUID(SDL_JoystickGUID guid); // Get the implementation name for a GUID
562 char * SDL_GameControllerMapping(SDL_GameController *gamecontroller); // Get the implementation name
563 SDL_bool SDL_IsGameController(int joystick_index); // Query whether a joystick is a game controller
564 const char * SDL_GameControllerNameForIndex(int joystick_index); // Get the implementation name
565 const char * SDL_GameControllerPathForIndex(int joystick_index); // Get the implementation path
566 SDL_GameControllerType SDL_GameControllerTypeForIndex(int joystick_index); // Get the game controller type
567 char * SDL_GameControllerMappingForDeviceIndex(int joystick_index); // Get the implementation name
568 SDL_GameController * SDL_GameControllerOpen(int joystick_index); // Open a game controller
569 SDL_GameController * SDL_GameControllerFromInstanceID(SDL_JoystickID joyid); // Get the SDL_GameController from an instance ID
570 SDL_GameController * SDL_GameControllerFromPlayerIndex(int player_index); // Get the SDL_GameController from a player index
571 const char * SDL_GameControllerName(SDL_GameController *gamecontroller); // Get the implementation name
572 const char * SDL_GameControllerPath(SDL_GameController *gamecontroller); // Get the implementation path
573 SDL_GameControllerType SDL_GameControllerGetType(SDL_GameController *gamecontroller); // Get the game controller type
574 int SDL_GameControllerGetPlayerIndex(SDL_GameController *gamecontroller); // Get the player index
575 void SDL_GameControllerSetPlayerIndex(SDL_GameController *gamecontroller, int player_index); // Set the player index
576 Uint16 SDL_GameControllerGetVendor(SDL_GameController *gamecontroller); // Get the USB vendor ID
577 Uint16 SDL_GameControllerGetProduct(SDL_GameController *gamecontroller); // Get the USB product ID
578 Uint16 SDL_GameControllerGetProductVersion(SDL_GameController *gamecontroller); // Get the product version
579 Uint16 SDL_GameControllerGetFirmwareVersion(SDL_GameController *gamecontroller); // Get the firmware version
580 const char * SDL_GameControllerGetSerial(SDL_GameController *gamecontroller); // Get the serial number
581 SDL_bool SDL_GameControllerGetAttached(SDL_GameController *gamecontroller); // Get the status of a game controller
582 SDL_Joystick * SDL_GameControllerGetJoystick(SDL_GameController *gamecontroller); // Get the joystick
583 int SDL_GameControllerEventState(int state); // Enable/disable game controller event processing
584 void SDL_GameControllerUpdate(void); // Update the current state of the game controllers
585 SDL_GameControllerAxis SDL_GameControllerGetAxisFromString(const char *str); // Get the axis from a string
586 const char * SDL_GameControllerGetStringForAxis(SDL_GameControllerAxis axis); // Get the string for an axis
587 SDL_GameControllerButtonBind SDL_GameControllerGetBindForAxis(SDL_GameController *gamecontroller, SDL_GameControllerAxis axis); // Get the button bind for an axis
588 SDL_bool SDL_GameControllerHasAxis(SDL_GameController *gamecontroller, SDL_GameControllerAxis axis); // Query whether a game controller has an axis
589 Sint16 SDL_GameControllerGetAxis(SDL_GameController *gamecontroller, SDL_GameControllerAxis axis); // Get the current axis value
590 SDL_GameControllerButton SDL_GameControllerGetButtonFromString(const char *str); // Get the button from a string
591 const char * SDL_GameControllerGetStringForButton(SDL_GameControllerButton button); // Get the string for a button
592 SDL_GameControllerButtonBind SDL_GameControllerGetBindForButton(SDL_GameController *gamecontroller, SDL_GameControllerButton button); // Get the button bind for a button
593 Uint8 SDL_GameControllerHasButton(SDL_GameController *gamecontroller, SDL_GameControllerButton button); // Query whether a game controller has a button
594 int SDL_GameControllerGetButton(SDL_GameController *gamecontroller, SDL_GameControllerButton button); // Get the current button value
595 int SDL_GameControllerGetNumTouchpads(SDL_GameController *gamecontroller); // Get the number of touchpads
596 int SDL_GameControllerGetNumTouchpadFingers(SDL_GameController *gamecontroller, int touchpad); // Get the number of fingers on a touchpad
597 float * SDL_GameControllerGetTouchpadFinger(SDL_GameController *gamecontroller, int touchpad, int finger, float *x, float *y, float *pressure); // Get the touchpad coordinates and pressure
598 SDL_bool SDL_GameControllerHasSensor(SDL_GameController *gamecontroller, SDL_SensorType type); // Query whether a game controller has a sensor
599 int SDL_GameControllerSetSensorEnabled(SDL_GameController *gamecontroller, SDL_SensorType type, SDL_bool enabled); // Enable/disable a sensor
600 float SDL_GameControllerIsSensorEnabled(SDL_GameController *gamecontroller, SDL_SensorType type); // Query whether a sensor is enabled
601 float SDL_GameControllerGetSensorDataRate(SDL_GameController *gamecontroller, SDL_SensorType type); // Get the sensor data rate
602 int SDL_GameControllerGetSensorData(SDL_GameController *gamecontroller, SDL_SensorType type, float *data, int num_values); // Get the sensor data
603 int SDL_GameControllerGetSensorDataWithTimestamp(SDL_GameController *gamecontroller, SDL_SensorType type, Uint64 timestamp, float *data, int num_values); // Get the sensor data with a timestamp
```

```
610 int SDL_GameControllerRumble (SDL_GameController *gamecontroller, Uint16 low_frequency_rumble,
611                               Uint16 high_frequency_rumble, Uint32 duration_ms);
612 int SDL_GameControllerRumbleTriggers (SDL_GameController *gamecontroller, Uint16 left_rumble, Uint16
613                                       Uint32 duration_ms);
614 SDL_bool SDL_GameControllerHasLED (SDL_GameController *gamecontroller);
615 SDL_bool SDL_GameControllerHasRumble (SDL_GameController *gamecontroller);
616 SDL_bool SDL_GameControllerHasRumbleTriggers (SDL_GameController *gamecontroller);
617 int SDL_GameControllerSetLED (SDL_GameController *gamecontroller, Uint8 red, Uint8 green, Uint8 blue);
618 int SDL_GameControllerSendEffect (SDL_GameController *gamecontroller, const void *data, int size);
619 void SDL_GameControllerClose (SDL_GameController *gamecontroller);
620 const char* SDL_GameControllerGetAppleSFSymbolsNameForButton (SDL_GameController *gamecontroller, SDL_GameControllerButton button);
621 const char* SDL_GameControllerGetAppleSFSymbolsNameForAxis (SDL_GameController *gamecontroller, SDL_GameControllerAxis axis);
622
623 ///-----
624 /// Sensors (SDL_sensor.h)
625 ///-----
626 void SDL_LockSensors (void); // Lock for multi-threaded access
627 void SDL_UnlockSensors (void); // Unlock for multi-threaded access
628 int SDL_NumSensors (void); // Count the number of sensors
629 const char* SDL_SensorGetDeviceName (int device_index); // Get the implementation dependent name
630 SDL_SensorType SDL_SensorGetDeviceType (int device_index); // Get the type of a sensor
631 int SDL_SensorGetDeviceNonPortableType (int device_index); // Get the platform specific type
632 SDL_SensorID SDL_SensorGetDeviceInstanceID (int device_index); // Get the instance ID
633 SDL_Sensor* SDL_SensorOpen (int device_index); // Open a sensor for use
634 SDL_Sensor* SDL_SensorFromInstanceID (SDL_SensorID instance_id); // Return the SDL_Sensor for a given instance ID
635 const char* SDL_SensorGetName (SDL_Sensor *sensor); // Get the implementation dependent name
636 SDL_SensorType SDL_SensorGetType (SDL_Sensor *sensor); // Get the type of a sensor
637 int SDL_SensorGetNonPortableType (SDL_Sensor *sensor); // Get the platform specific type
638 SDL_SensorID SDL_SensorGetInstanceID (SDL_Sensor *sensor); // Get the instance ID
639 int SDL_SensorGetData (SDL_Sensor *sensor, float *data, int num_values); // Get the current sensor data
640 int SDL_SensorGetDataWithTimestamp (SDL_Sensor *sensor, Uint64 *timestamp, float *data, int num_values); // Get the current sensor data and timestamp
641 void SDL_SensorClose (SDL_Sensor *sensor); // Close a sensor
642 void SDL_SensorUpdate (void); // Update the current sensor data
643
644 ///-----
645 /// Force Feedback Support (SDL_haptic.h)
646 ///-----
647 int SDL_NumHaptics (void); // Count the number of haptic devices attached
648 const char* SDL_HapticName (int device_index); // Get the implementation dependent name
649 SDL_Haptic* SDL_HapticOpen (int device_index); // Open a haptic device for use
650 int SDL_HapticOpened (int device_index); // Check if the haptic device at the device index is open
651 int SDL_HapticIndex (SDL_Haptic *haptic); // Get the index of a haptic device
652 int SDL_MouseIsHaptic (void); // Query whether or not the current mouse is a haptic device
653 SDL_Haptic* SDL_HapticOpenFromMouse (void); // Try to open a haptic device from the current mouse
654 int SDL_JoystickIsHaptic (SDL_Joystick *joystick); // Query if a joystick has haptic features
655 SDL_Haptic* SDL_HapticOpenFromJoystick (SDL_Joystick *joystick); // Open a haptic device for use from a joystick
656 void SDL_HapticClose (SDL_Haptic *haptic); // Close a haptic device previously opened
657 int SDL_HapticNumEffects (SDL_Haptic *haptic); // Get the number of effects a haptic device supports
658 int SDL_HapticNumEffectsPlaying (SDL_Haptic *haptic); // Get the number of effects a haptic device currently has playing
659 unsigned int SDL_HapticQuery (SDL_Haptic *haptic); // Get the haptic device's supported features
660 int SDL_HapticNumAxes (SDL_Haptic *haptic); // Get the number of haptic axes the device supports
661 int SDL_HapticEffectSupported (SDL_Haptic *haptic, SDL_HapticEffect *effect); // Check to see if an effect is supported
662 int SDL_HapticNewEffect (SDL_Haptic *haptic, SDL_HapticEffect *effect); // Create a new haptic effect on a specific device
663 int SDL_HapticUpdateEffect (SDL_Haptic *haptic, int effect, SDL_HapticEffect *data); // Update the properties of an effect
664 int SDL_HapticRunEffect (SDL_Haptic *haptic, int effect, Uint32 iterations); // Run the haptic effect on its associated device
665 int SDL_HapticStopEffect (SDL_Haptic *haptic, int effect); // Stop the haptic effect on its associated device
666 void SDL_HapticDestroyEffect (SDL_Haptic *haptic, int effect); // Destroy a haptic effect on the device
667 int SDL_HapticGetEffectStatus (SDL_Haptic *haptic, int effect); // Get the status of the current effect
668 int SDL_HapticSetGain (SDL_Haptic *haptic, int gain); // Set the global gain of the specified haptic device
669 int SDL_HapticSetAutocenter (SDL_Haptic *haptic, int autocenter); // Set the global autocenter of the device
670 int SDL_HapticPause (SDL_Haptic *haptic); // Pause a haptic device
671 int SDL_HapticUnpause (SDL_Haptic *haptic); // Unpause a haptic device
672 int SDL_HapticStopAll (SDL_Haptic *haptic); // Stop all the currently playing effects
673 int SDL_HapticRumbleSupported (SDL_Haptic *haptic); // Check whether rumble is supported on a haptic device
674 int SDL_HapticRumbleInit (SDL_Haptic *haptic); // Initialize a haptic device for simple rumble
675 int SDL_HapticRumblePlay (SDL_Haptic *haptic, float strength, Uint32 length); // Run a simple rumble effect on a haptic device
676 int SDL_HapticRumbleStop (SDL_Haptic *haptic); // Stop the simple rumble on a haptic device
677
678
679 // AUDIO
680
681
682
683
684
685
686 ///-----
687 /// Audio Device Management, Playing and Recording (SDL_audio.h)
688 ///-----
689 int SDL_GetNumAudioDrivers (void); // Use this function to get the number of audio drivers
690 const char* SDL_GetAudioDriver (int index); // Use this function to get the name of an audio driver
691 int SDL_AudioInit (const char *driver_name); // Use this function to initialize the audio
692 void SDL_AudioQuit (void); // Use this function to shut down the audio
693 const char* SDL_GetCurrentAudioDriver (void); // Get the name of the current audio driver
694 int SDL_OpenAudio (SDL_AudioSpec *desired, SDL_AudioSpec *obtained); // This function opens the audio device
695 int SDL_GetNumAudioDevices (int iscapture); // Get the number of audio devices
696 const char* SDL_GetAudioDeviceName (int index, int iscapture); // Get the human readable name of an audio device
697 int SDL_GetAudioDeviceSpec (int index, int iscapture, SDL_AudioSpec *spec); // Get the preferred audio format for a device
698 int SDL_GetDefaultAudioInfo (char **name, SDL_AudioSpec *spec, int iscapture); // Get the name and spec of the default audio device
699 SDL_AudioDeviceID SDL_OpenAudioDevice (const char *device, int iscapture, const SDL_AudioSpec *desired,
700                                       SDL_AudioSpec *obtained, int allowed_changes); // Open a specific audio device
701 int SDL_GetAudioStatus (void); // This function returns the status of the audio device
702 int SDL_GetAudioDeviceStatus (SDL_AudioDeviceID dev); // This function returns the status of a specific audio device
703 void SDL_PauseAudio (int pause_on); // Use this function to pause the audio
704 void SDL_PauseAudioDevice (SDL_AudioDeviceID dev, int pause_on); // Use this function to pause a specific audio device
705 SDL_AudioSpec* SDL_LoadWAV_RW (SDL_RWops *src, int freesrc, SDL_AudioSpec *spec, Uint8 **audio_buf,
706                               Uint32 *audio_len); // Load a WAV file from a source
707 void SDL_FreeWAV (Uint8 *audio_buf); // Free data pointed to by audio_buf
708 int SDL_BuildAudioCVT (SDL_AudioCVT *cvt, SDL_AudioFormat src_format, Uint8 src_channels, int src_rate,
709                      SDL_AudioFormat dst_format, Uint8 dst_channels, int dst_rate); // Initialize an audio conversion structure
710 int SDL_ConvertAudio (SDL_AudioCVT *cvt); // Convert audio data
711 SDL_AudioStream* SDL_NewAudioStream (const SDL_AudioFormat src_format, const Uint8 src_channels, const int src_rate,
712                                     const SDL_AudioFormat dst_format, const Uint8 dst_channels, const int dst_rate); // Create a new audio stream
713 int SDL_AudioStreamPut (SDL_AudioStream *stream, const void *buf, int len); // Add data to the audio stream
714 int SDL_AudioStreamGet (SDL_AudioStream *stream, void *buf, int len); // Get converted audio data
715 int SDL_AudioStreamAvailable (SDL_AudioStream *stream); // Get the number of bytes available for writing
716 int SDL_AudioStreamFlush (SDL_AudioStream *stream); // Tell the stream to flush
717 void SDL_AudioStreamClear (SDL_AudioStream *stream); // Clear any pending data
718 void SDL_FreeAudioStream (SDL_AudioStream *stream); // Free an audio stream
719 void SDL_MixAudio (Uint8 *dst, const Uint8 *src, Uint32 len, int volume); // This function mixes audio data
720 void SDL_MixAudioFormat (Uint8 *dst, const Uint8 *src, SDL_AudioFormat format, Uint32 len, int volume); // Mix audio data in a specific format
721 int SDL_QueueAudio (SDL_AudioDeviceID dev, const void *data, Uint32 len); // Queue more audio data for playing
722 Uint32 SDL_DequeueAudio (SDL_AudioDeviceID dev, void *data, Uint32 len); // Dequeue more audio data
723 Uint32 SDL_GetQueuedAudioSize (SDL_AudioDeviceID dev); // Get the number of bytes of audio data queued for playing
724 void SDL_ClearQueuedAudio (SDL_AudioDeviceID dev); // Drop any queued audio data
725 void SDL_LockAudio (void); // This function locks the audio device
726 void SDL_LockAudioDevice (SDL_AudioDeviceID dev); // Use this function to lock a specific audio device
727 void SDL_UnlockAudio (void); // This function unlocks the audio device
728 void SDL_UnlockAudioDevice (SDL_AudioDeviceID dev); // Use this function to unlock a specific audio device
729 void SDL_CloseAudio (void); // This function closes the audio device
730 void SDL_CloseAudioDevice (SDL_AudioDeviceID dev); // Use this function to close a specific audio device
731
732
733 // THEFANS
734
735
736
```

```
737 //
738 //
739
740 ///-----
741 /// Thread Management (SDL_thread.h)
742 ///-----
743 SDL_Thread * SDL_CreateThread (SDL_ThreadFunction fn, const char *name, void *data); // Create a new th
744 SDL_Thread * SDL_CreateThreadWithStackSize (SDL_ThreadFunction fn, const char *name, const size_t stacksize, void *data); // Create a new th
745 const char * SDL_GetThreadName (SDL_Thread *thread); // Get the thread
746 SDL_threadID SDL_GetThreadID (void); // Get the thread
747 SDL_threadID SDL_GetThreadID (SDL_Thread *thread); // Get the thread
748 int SDL_SetThreadPriority (SDL_ThreadPriority priority); // Set the priorit
749 void SDL_WaitThread (SDL_Thread *thread, int *status); // Wait for a thre
750 void SDL_DetachThread (SDL_Thread *thread); // Let a thread cl
751 SDL_TLSID SDL_TLSCreate (void); // Create a piece
752 void * SDL_TLSGet (SDL_TLSID id); // Get the current
753 int SDL_TLSSet (SDL_TLSID id, const void *value, void (*destructor)(void*)); // Set the current
754 void SDL_TLSCleanup (void); // Cleanup all TLS
755
756 ///-----
757 /// Thread Synchronization Primitives (SDL_mutex.h)
758 ///-----
759 SDL_mutex * SDL_CreateMutex (void); // Create a new mutex.
760 int SDL_LockMutex (SDL_mutex *mutex) SDL_ACQUIRE(mutex); // Lock the mutex.
761 int SDL_TryLockMutex (SDL_mutex *mutex) SDL_TRY_ACQUIRE(0, mutex); // Try to lock a mutex without blocking.
762 int SDL_UnlockMutex (SDL_mutex *mutex) SDL_RELEASE(mutex); // Unlock the mutex.
763 void SDL_DestroyMutex (SDL_mutex *mutex); // Destroy a mutex created with SDL_CreateMutex().
764 SDL_sem * SDL_CreateSemaphore (Uint32 initial_value); // Create a semaphore.
765 void SDL_DestroySemaphore (SDL_sem *sem); // Destroy a semaphore.
766 int SDL_SemWait (SDL_sem *sem); // Wait until a semaphore has a positive value and then decre
767 int SDL_SemTryWait (SDL_sem *sem); // See if a semaphore has a positive value and decrement it
768 int SDL_SemWaitTimeout (SDL_sem *sem, Uint32 timeout); // Wait until a semaphore has a positive value and then decre
769 int SDL_SemPost (SDL_sem *sem); // Atomically increment a semaphore's value and wake waiting
770 Uint32 SDL_SemValue (SDL_sem *sem); // Get the current value of a semaphore.
771 SDL_cond * SDL_CreateCond (void); // Create a condition variable.
772 void SDL_DestroyCond (SDL_cond *cond); // Destroy a condition variable.
773 int SDL_CondSignal (SDL_cond *cond); // Restart one of the threads that are waiting on the condit
774 int SDL_CondBroadcast (SDL_cond *cond); // Restart all threads that are waiting on the condition var
775 int SDL_CondWait (SDL_cond *cond, SDL_mutex *mutex); // Wait until a condition variable is signaled.
776 int SDL_CondWaitTimeout (SDL_cond *cond, SDL_mutex *mutex, Uint32 ms); // Wait until a condition variable is signaled or a certain
777
778 ///-----
779 /// Atomic Operations (SDL_atomic.h)
780 ///-----
781 SDL_bool SDL_AtomicTryLock (SDL_SpinLock *lock); // Try to lock a spin lock by setting it to a non-zero value.
782 void SDL_AtomicLock (SDL_SpinLock *lock); // Lock a spin lock by setting it to a non-zero value.
783 void SDL_AtomicUnlock (SDL_SpinLock *lock); // Unlock a spin lock by setting it to 0.
784 void SDL_MemoryBarrierAcquire (void); // Release a memory barrier
785 void SDL_MemoryBarrierRelease (void); // Acquire a memory barrier
786 void SDL_CPUPauseInstruction (void); // Execute hardware pause instruction, if supported
787 SDL_bool SDL_AtomicCAS (SDL_atomic_t *a, int oldval, int newval); // Set an atomic variable to a new value if it is currently an
788 int SDL_AtomicSet (SDL_atomic_t *a, int v); // Set an atomic variable to a value.
789 int SDL_AtomicGet (SDL_atomic_t *a); // Get the value of an atomic variable.
790 int SDL_AtomicAdd (SDL_atomic_t *a, int v); // Add to an atomic variable.
791 SDL_bool SDL_AtomicCASPtr (void **a, void *oldval, void *newval); // Set a pointer to a new value if it is currently an old value
792 void * SDL_AtomicSetPtr (void **a, void *v); // Set a pointer to a value atomically.
793 void * SDL_AtomicGetPtr (void **a); // Get the value of a pointer atomically.
794
795
796 // TIME
797
798
799
800
801
802
803 ///-----
804 /// Timer Support (SDL_timer.h)
805 ///-----
806 Uint64 SDL_GetTicks64 (void); // Get the number of milliseconds since
807 Uint32 SDL_GetTicks (void); // Get the number of milliseconds since
808 Uint64 SDL_GetPerformanceCounter (void); // Get the current value of the high res
809 Uint64 SDL_GetPerformanceFrequency (void); // Get the count per second of the high
810 void SDL_Delay (Uint32 ms); // Wait a specified number of millisecon
811 SDL_TimerID SDL_AddTimer (Uint32 interval, SDL_TimerCallback callback, void *param); // Call a callback function at a future
812 SDL_bool SDL_RemoveTimer (SDL_TimerID id); // Remove a timer created with SDL_AddT
813
814
815 // FILE I/O
816
817
818
819
820
821
822 ///-----
823 /// Filesystem Paths (SDL_filesystem.h)
824 ///-----
825 char * SDL_GetBasePath (void); // Get the directory where the application was run from.
826 char * SDL_GetPrefPath (const char *org, const char *app); // Get the user-and-app-specific path where files can be written.
827
828 ///-----
829 /// File I/O Abstraction (SDL_rwops.h)
830 ///-----
831 SDL_RWops * SDL_RWFromFile (const char *file, const char *mode); // Use this function to create a new SDL_RWop
832 SDL_RWops * SDL_RWFromFP (void *fptr, SDL_bool autoclose); // Use this function to create an SDL_RWops s
833 SDL_RWops * SDL_RWFromMem (void *mem, int size); // Use this function to prepare a read-write
834 SDL_RWops * SDL_RWFromConstMem (const void *mem, int size); // Use this function to prepare a read-only m
835 SDL_RWops * SDL_AllocRW (void); // Use this function to allocate an empty, un
836 void SDL_FreeRW (SDL_RWops *area); // Use this function to free an SDL_RWops str
837 Sint64 SDL_RWsize (SDL_RWops *context); // Use this function to get the size of the d
838 Sint64 SDL_RWseek (SDL_RWops *context, Sint64 offset, int whence); // Seek within an SDL_RWops data stream.
839 Sint64 SDL_RWtell (SDL_RWops *context); // Determine the current read/write offset in
840 size_t SDL_RWread (SDL_RWops *context, void *ptr, size_t size, size_t maxnum); // Read from a data source.
841 size_t SDL_RWwrite (SDL_RWops *context, const void *ptr, size_t size, size_t num); // Write to an SDL_RWops data stream.
842 int SDL_RWclose (SDL_RWops *context); // Close and free an allocated SDL_RWops stru
843 void * SDL_LoadFile_RW (SDL_RWops *src, size_t *datasize, int freesrc); // Load all the data from an SDL data stream.
844 void * SDL_LoadFile (const char *file, size_t *datasize); // Load all the data from a file path.
845 Uint8 SDL_ReadU8 (SDL_RWops *src); // Use this function to read a byte from an S
846 Uint16 SDL_ReadLE16 (SDL_RWops *src); // Use this function to read 16 bits of littl
847 Uint16 SDL_ReadBE16 (SDL_RWops *src); // Use this function to read 16 bits of big-e
848 Uint32 SDL_ReadLE32 (SDL_RWops *src); // Use this function to read 32 bits of littl
849 Uint32 SDL_ReadBE32 (SDL_RWops *src); // Use this function to read 32 bits of big-e
850 Uint64 SDL_ReadLE64 (SDL_RWops *src); // Use this function to read 64 bits of littl
851 Uint64 SDL_ReadBE64 (SDL_RWops *src); // Use this function to read 64 bits of big-e
852 size_t SDL_WriteU8 (SDL_RWops *dst, Uint8 value); // Use this function to write a byte to an SD
853 size_t SDL_WriteLE16 (SDL_RWops *dst, Uint16 value); // Use this function to write 16 bits in nati
854 size_t SDL_WriteBE16 (SDL_RWops *dst, Uint16 value); // Use this function to write 16 bits in nati
855 size_t SDL_WriteLE32 (SDL_RWops *dst, Uint32 value); // Use this function to write 32 bits in nati
856 size_t SDL_WriteBE32 (SDL_RWops *dst, Uint32 value); // Use this function to write 32 bits in nati
857 size_t SDL_WriteBE64 (SDL_RWops *dst, Uint64 value); // Use this function to write 64 bits in nati
858 size_t SDL_WriteLE64 (SDL_RWops *dst, Uint64 value); // Use this function to write 64 bits in nati
859
860
861 // SUBBER AB IEFTE I NI I
862
863
```

```
864 //-----
865 //
866 //
867 //
868 //-----
869 // Shared Object Loading and Function Lookup (SDL_loadso.h)
870 //-----
871 void * SDL_LoadObject (const char *sofile); // Dynamically load a shared object.
872 void * SDL_LoadFunction (void *handle, const char *name); // Look up the address of the named function in a shared object.
873 void SDL_UnloadObject (void *handle); // Unload a shared object from memory.
874
875
876 // PLATFORM / CPU
877 //-----
878 // Platform Detection (SDL_platform.h)
879 //-----
880 const char * SDL_GetPlatform (void); // Get the name of the platform.
881
882 //-----
883 // CPU Feature Detection (SDL_cpuinfo.h)
884 //-----
885
886
887
888
889 int SDL_GetCPUCount (void); // Get the number of CPU cores available.
890 int SDL_GetCPUCacheLineSize (void); // Determine the L1 cache line size of the CPU.
891 SDL_bool SDL_HasRDTSC (void); // Determine whether the CPU has the RDTSC instruction.
892 SDL_bool SDL_HasAltivec (void); // Determine whether the CPU has Altivec features.
893 SDL_bool SDL_HasMMX (void); // Determine whether the CPU has MMX features.
894 SDL_bool SDL_Has3DNow (void); // Determine whether the CPU has 3DNow! features.
895 SDL_bool SDL_HasSSE (void); // Determine whether the CPU has SSE features.
896 SDL_bool SDL_HasSSE2 (void); // Determine whether the CPU has SSE2 features.
897 SDL_bool SDL_HasSSE3 (void); // Determine whether the CPU has SSE3 features.
898 SDL_bool SDL_HasSSE41 (void); // Determine whether the CPU has SSE4.1 features.
899 SDL_bool SDL_HasSSE42 (void); // Determine whether the CPU has SSE4.2 features.
900 SDL_bool SDL_HasAVX (void); // Determine whether the CPU has AVX features.
901 SDL_bool SDL_HasAVX2 (void); // Determine whether the CPU has AVX2 features.
902 SDL_bool SDL_HasAVX512F (void); // Determine whether the CPU has AVX-512F (foundation) features.
903 SDL_bool SDL_HasARMSIMD (void); // Determine whether the CPU has ARM SIMD (ARMv6) features.
904 SDL_bool SDL_HasNEON (void); // Determine whether the CPU has NEON (ARM SIMD) features.
905 SDL_bool SDL_HasLSX (void); // Determine whether the CPU has LSX (LOONGARCH SIMD) features.
906 SDL_bool SDL_HasLASX (void); // Determine whether the CPU has LASX (LOONGARCH SIMD) features.
907 int SDL_GetSystemRAM (void); // Get the amount of RAM configured in the system.
908 size_t SDL_SIMDGetAlignment (void); // Report the alignment this system needs for SIMD allocations.
909 void * SDL_SIMDAlloc (const size_t len); // Allocate memory in a SIMD-friendly way.
910 void * SDL_SIMDRealloc (void *mem, const size_t len); // Reallocate memory obtained from SDL_SIMDAlloc
911 void SDL_SIMDFree (void *ptr); // Deallocate memory obtained from SDL_SIMDAlloc
912
913 //-----
914 // Byte Order and Byte Swapping (SDL_endian.h)
915 //-----
916
917
918 Uint16 SDL_Swap16 (Uint16 x); // Unconditionally byte swap the provided data.
919 Uint32 SDL_Swap32 (Uint32 x); // Unconditionally byte swap the provided data.
920 Uint64 SDL_Swap64 (Uint64 x); // Unconditionally byte swap the provided data.
921 float SDL_SwapFloat (float x); // Unconditionally byte swap the provided data.
922
923
924 Uint16 SDL_SwapLE16 (Uint16 X); // Byteswap item from the specified endianness to the native endianness if necessary.
925 Uint32 SDL_SwapLE32 (Uint32 X); // Byteswap item from the specified endianness to the native endianness if necessary.
926 Uint64 SDL_SwapLE64 (Uint64 X); // Byteswap item from the specified endianness to the native endianness if necessary.
927 float SDL_SwapFloatLE (float X); // Byteswap item from the specified endianness to the native endianness if necessary.
928 Uint16 SDL_SwapBE16 (Uint16 X); // Byteswap item from the specified endianness to the native endianness if necessary.
929 Uint32 SDL_SwapBE32 (Uint32 X); // Byteswap item from the specified endianness to the native endianness if necessary.
930 Uint64 SDL_SwapBE64 (Uint64 X); // Byteswap item from the specified endianness to the native endianness if necessary.
931 float SDL_SwapFloatBE (float X); // Byteswap item from the specified endianness to the native endianness if necessary.
932
933 //-----
934 // Bit Manipulation (SDL_bits.h)
935 //-----
936
937
938
939 int SDL_MostSignificantBitIndex32 (Uint32 x); // Get the index of the most significant bit. Undefined when called with 0.
940 SDL_bool SDL_HasExactlyOneBitSet32 (Uint32 x); // Returns true if integer has exactly one bit set.
941
942
943 // POWER MANAGEMENT
944 //-----
945 // Power Management Status (SDL_power.h)
946 //-----
947
948
949 SDL_PowerState SDL_GetPowerInfo (int *seconds, int *percent); // Get the current power supply details.
950
951
952 // ADDITIONAL FUNCTION
953 //-----
954 // Platform-specific Functionality (SDL_system.h)
955 //-----
956
957
958 //-----
959 // Windows
960 //-----
961
962
963
964
965 void SDL_SetWindowsMessageHook (SDL_WindowsMessageHook callback, void *userdata); // Set a callback for every Windows
966 int SDL_Direct3D9GetAdapterIndex (int displayIndex); // Get the D3D9 adapter index that m
967 IDirect3DDevice9* SDL_RendererGetD3D9Device (SDL_Renderer *renderer); // Get the D3D9 device associated w
968 ID3D11Device* SDL_RendererGetD3D11Device (SDL_Renderer *renderer); // Get the D3D11 device associated w
969 ID3D12Device* SDL_RendererGetD3D12Device (SDL_Renderer *renderer); // Get the D3D12 device associated w
970 SDL_bool SDL_DXGIGetOutputInfo (int displayIndex, int *adapterIndex, int *outputIndex); // Get the DXGI Adapter and Output i
971 int SDL_GDKGetTaskQueue (XTaskQueueHandle *outTaskQueue); // Gets a reference to the global as
972
973 //-----
974 // Linux
975 //-----
976
977
978
979
980 int SDL_LinuxSetThreadPriority (Sint64 threadID, int priority); // Sets the UNIX nice value for a thread.
981 int SDL_LinuxSetThreadPriorityAndPolicy (Sint64 threadID, int sdlPriority, int schedPolicy); // Sets the priority (not nice level) and sched
982
983 //-----
984 // iOS
985 //-----
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
```



```
991
992 // Android
993 //-----
994 void *      SDL_AndroidGetJNIEnv          (void); // Get the Android Java Native Interface Environment of the current thread.
995 void *      SDL_AndroidGetActivity        (void); // Retrieve the Java instance of the Android activity class.
996 int         SDL_GetAndroidSDKVersion      (void); // Query Android API level of the current device.
997 SDL_bool    SDL_IsAndroidTV               (void); // Query if the application is running on Android TV.
998 SDL_bool    SDL_IsChromebook              (void); // Query if the application is running on a Chromebook.
999 SDL_bool    SDL_IsDeXMode                 (void); // Query if the application is running on a Samsung DeX docking station.
1000 void        SDL_AndroidBackButton         (void); // Trigger the Android system back button behavior.
1001 const char * SDL_AndroidGetInternalStoragePath (void); // Get the path used for internal storage for this application.
1002 int         SDL_AndroidGetExternalStorageState (void); // Get the current state of external storage.
1003 const char * SDL_AndroidGetExternalStoragePath (void); // Get the path used for external storage for this application.
1004
1005 SDL_bool    SDL_AndroidRequestPermission   (const char *permission); // Request permissions at runtime.
1006 int         SDL_AndroidShowToast          (const char *message, int duration, // Shows an Android toast notification.
1007                                           int gravity, int xoffset, int yoffset); // Send a user command to SDLActivity.
1008 int         SDL_AndroidSendMessage        (Uint32 command, int param);
1009
1010 // WinRT (Windows Phone)
1011 //-----
1012 const wchar_t * SDL_WinRTGetFSPathUNICODE (SDL_WinRT_Path pathType); // Retrieve a WinRT defined path on the local file system.
1013 const char *     SDL_WinRTGetFSPathUTF8    (SDL_WinRT_Path pathType); // Retrieve a WinRT defined path on the local file system.
1014 SDL_WinRT_DeviceFamily SDL_WinRTGetDeviceFamily (void); // Detects the device family of WinRT platform at runtime.
1015
1016 // Misc.
1017 //-----
1018 SDL_bool SDL_IsTablet (void); // Query if the current device is a tablet.
1019
1020 ///-----
1021 /// Standard Library Functionality (SDL_stdinc.h)
1022 ///-----
1023
1024 // Memory
1025 //-----
1026 void *      SDL_malloc (size_t size);
1027 void *      SDL_calloc (size_t nmemb, size_t size);
1028 void *      SDL_realloc (void *mem, size_t size);
1029 void        SDL_free (void *mem);
1030 void        SDL_GetOriginalMemoryFunctions (SDL_malloc_func *malloc_func, SDL_calloc_func *calloc_func, // Get the original set of SDL memory functions.
1031                                             SDL_realloc_func *realloc_func, SDL_free_func *free_func);
1032 void        SDL_GetMemoryFunctions (SDL_malloc_func *malloc_func, SDL_calloc_func *calloc_func, // Get the current set of SDL memory functions.
1033                                     SDL_realloc_func *realloc_func, SDL_free_func *free_func);
1034 int         SDL_SetMemoryFunctions (SDL_malloc_func malloc_func, SDL_calloc_func calloc_func, // Replace SDL's memory allocation functions.
1035                                     SDL_realloc_func realloc_func, SDL_free_func free_func); // Get the number of outstanding (unfreed)
1036 int         SDL_GetNumAllocations (void);
1037
1038 // Environment Variables
1039 //-----
1040 char *      SDL_getenv (const char *name);
1041 int         SDL_setenv (const char *name, const char *value, int overwrite);
1042
1043 // Sort/search
1044 //-----
1045 void        SDL_qsort (void *base, size_t nmemb, size_t size, int (*compare) (const void *, const void *));
1046 void *      SDL_bsearch (const void *key, const void *base, size_t nmemb, size_t size, int (*compare) (const void *, const void *));
1047
1048 // Strings
1049 //-----
1050 int         SDL_isalpha (int x);
1051 int         SDL_isalnum (int x);
1052 int         SDL_isblank (int x);
1053 int         SDL_iscntrl (int x);
1054 int         SDL_isdigit (int x);
1055 int         SDL_isxdigit (int x);
1056 int         SDL_isspace (int x);
1057 int         SDL_isupper (int x);
1058 int         SDL_islower (int x);
1059 int         SDL_isprint (int x);
1060 int         SDL_isgraph (int x);
1061 int         SDL_toupper (int x);
1062 int         SDL_tolower (int x);
1063 Uint16      SDL_crc16 (Uint16 crc, const void *data, size_t len);
1064 Uint32      SDL_crc32 (Uint32 crc, const void *data, size_t len);
1065 void *      SDL_memset (void *dst, int c, size_t len);
1066 void *      SDL_memcpy (SDL_OUT_BYTECAP(len) void *dst, SDL_IN_BYTECAP(len) const void *src, size_t len);
1067 void *      SDL_memmove (SDL_OUT_BYTECAP(dwords*4) void *dst, SDL_IN_BYTECAP(dwords*4) const void *src, size_t dwords);
1068 void *      SDL_memmove (SDL_OUT_BYTECAP(len) void *dst, SDL_IN_BYTECAP(len) const void *src, size_t len);
1069 int         SDL_memcmp (const void *s1, const void *s2, size_t len);
1070 size_t      SDL_wcslen (const wchar_t *wstr);
1071 size_t      SDL_wcslcpy (SDL_OUT_Z_CAP(maxlen) wchar_t *dst, const wchar_t *src, size_t maxlen);
1072 size_t      SDL_wcslcat (SDL_INOUT_Z_CAP(maxlen) wchar_t *dst, const wchar_t *src, size_t maxlen);
1073 wchar_t *   SDL_wcsdup (const wchar_t *wstr);
1074 wchar_t *   SDL_wcsstr (const wchar_t *haystack, const wchar_t *needle);
1075 int         SDL_wcsncmp (const wchar_t *str1, const wchar_t *str2, size_t maxlen);
1076 int         SDL_wscasecmp (const wchar_t *str1, const wchar_t *str2);
1077 int         SDL_wscasecmp (const wchar_t *str1, const wchar_t *str2, size_t len);
1078 int         SDL_strlen (const char *str);
1079 size_t      SDL_strlen (SDL_OUT_Z_CAP(maxlen) char *dst, const char *src, size_t maxlen);
1080 size_t      SDL_utf8strlcpy (SDL_OUT_Z_CAP(dst_bytes) char *dst, const char *src, size_t dst_bytes);
1081 size_t      SDL_strlcat (SDL_INOUT_Z_CAP(maxlen) char *dst, const char *src, size_t maxlen);
1082 char *      SDL_strdup (const char *str);
1083 char *      SDL_strrev (char *str);
1084 char *      SDL_strupr (char *str);
1085 char *      SDL_strlwr (char *str);
1086 char *      SDL_strchr (const char *str, int c);
1087 char *      SDL_strrchr (const char *str, int c);
1088 char *      SDL_strstr (const char *haystack, const char *needle);
1089 char *      SDL_strcasestr (const char *haystack, const char *needle);
1090 char *      SDL_strtokr (char *s1, const char *s2, char **saveptr);
1091 size_t      SDL_utf8strlen (const char *str);
1092 size_t      SDL_utf8strnlen (const char *str, size_t bytes);
1093 char *      SDL_itoa (int value, char *str, int radix);
1094 char *      SDL_vitoa (unsigned int value, char *str, int radix);
1095 char *      SDL_ltoa (long value, char *str, int radix);
1096 char *      SDL_vltoa (unsigned long value, char *str, int radix);
1097 char *      SDL_lltoa ( Sint64 value, char *str, int radix);
1098 char *      SDL_vlltoa ( Uint64 value, char *str, int radix);
1099 int         SDL_atoi (const char *str);
1100 double      SDL_atof (const char *str);
1101 long        SDL_strtol (const char *str, char **endp, int base);
1102 unsigned long SDL_strtoul (const char *str, char **endp, int base);
1103 Sint64      SDL_strtoll (const char *str, char **endp, int base);
1104 Uint64      SDL_strtoull (const char *str, char **endp, int base);
1105 double      SDL_strtod (const char *str, char **endp);
1106 int         SDL_strcmp (const char *str1, const char *str2);
1107 int         SDL_strncmp (const char *str1, const char *str2, size_t maxlen);
1108 int         SDL_strcasecmp (const char *str1, const char *str2);
1109 int         SDL_strncasecmp (const char *str1, const char *str2, size_t len);
1110 int         SDL_sscanf (const char *text, SDL_SCANF_FORMAT_STRING const char *fmt, ...);
1111 int         SDL_vsscanf (const char *text, const char *fmt, va_list ap);
1112 int         SDL_snprintf (SDL_OUT_Z_CAP(maxlen) char *text, size_t maxlen, SDL_PRINTF_FORMAT_STRING const char *fmt, ...);
1113 int         SDL_vsnprintf (SDL_OUT_Z_CAP(maxlen) char *text, size_t maxlen, const char *fmt, va_list ap);
1114 int         SDL_asprintf (char **strp, SDL_PRINTF_FORMAT_STRING const char *fmt, ...);
1115 int         SDL_vasprintf (char **strp, const char *fmt, va_list ap);
```

```
1118
1119 // Math
1120 //-----
1121 int      SDL_abs      (int x);
1122 double   SDL_acos     (double x);
1123 float    SDL_acosf    (float x);
1124 double   SDL_asin     (double x);
1125 float    SDL_asinf    (float x);
1126 double   SDL_atan     (double x);
1127 float    SDL_atanf    (float x);
1128 double   SDL_atan2    (double y, double x);
1129 float    SDL_atan2f   (float y, float x);
1130 double   SDL_ceil     (double x);
1131 float    SDL_ceilf    (float x);
1132 double   SDL_copysign (double x, double y);
1133 float    SDL_copysignf (float x, float y);
1134 double   SDL_cos      (double x);
1135 float    SDL_cosf     (float x);
1136 double   SDL_exp      (double x);
1137 float    SDL_expf     (float x);
1138 double   SDL_fabs     (double x);
1139 float    SDL_fabsf    (float x);
1140 double   SDL_floor    (double x);
1141 float    SDL_floorf   (float x);
1142 double   SDL_trunc    (double x);
1143 float    SDL_truncf   (float x);
1144 double   SDL_fmod     (double x, double y);
1145 float    SDL_fmodf    (float x, float y);
1146 double   SDL_log      (double x);
1147 float    SDL_logf     (float x);
1148 double   SDL_log10    (double x);
1149 float    SDL_log10f   (float x);
1150 double   SDL_pow      (double x, double y);
1151 float    SDL_powf     (float x, float y);
1152 double   SDL_round    (double x);
1153 float    SDL_roundf   (float x);
1154 long     SDL_lround   (double x);
1155 long     SDL_lroundf  (float x);
1156 double   SDL_scalbn   (double x, int n);
1157 float    SDL_scalbnf  (float x, int n);
1158 double   SDL_sin      (double x);
1159 float    SDL_sinf     (float x);
1160 double   SDL_sqrt     (double x);
1161 float    SDL_sqrtf    (float x);
1162 double   SDL_tan      (double x);
1163 float    SDL_tanf     (float x);
1164
1165 // Unicode
1166 //-----
1167 SDL_iconv_t SDL_iconv_open (const char *tocode, const char *fromcode);
1168 int          SDL_iconv_close (SDL_iconv_t cd);
1169 size_t      SDL_iconv      (SDL_iconv_t cd, const char **inbuf, size_t *inbytesleft, char **outbuf, size_t *outbytesleft);
1170 char *      SDL_iconv_string (const char *tocode, const char *fromcode, const char *inbuf, size_t inbytesleft); // This function converts
1171 char *      SDL_iconv_utf8_locale (const char *inbuf);
1172 char *      SDL_iconv_utf8_ucs2   (const char *inbuf);
1173 char *      SDL_iconv_utf8_ucs4   (const char *inbuf);
1174 char *      SDL_iconv_wchar_utf8  (const char *inbuf);
1175 int         SDL_size_mul_overflow (size_t a, size_t b, size_t *ret); // If a * b would overflow, return -1. Otherwise store a * b via ret and
1176 int         SDL_size_add_overflow (size_t a, size_t b, size_t *ret); // If a + b would overflow, return -1. Otherwise store a + b via ret and
```

#sdl #cheatsheet

[◀ Auto-Tiling](#)[Recursion and Code Compression ▶](#)

0 reactions



0 comments

Write

Preview

Aa

Sign in to comment

143

Sign in with GitHub



© 2019 - 2025 Dan Bechard