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:

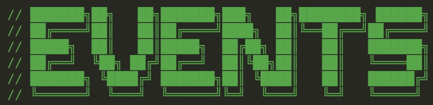
sdl2_quickref.c

Code

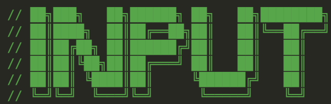
```c
// NOTE: For best viewing experience, click "Raw" in the top right of the Gist page, or download
// this file and view it in your favorite text editor with syntax highlighting!

//==============================================
//| Title        : SDL 2.0 API Quick Reference |
//| Author       : https://github.com/dbechrd/ |
//| Last updated : Jan 8, 2023                 |
//==============================================

// Based on: https://wiki.libsdl.org/SDL2/APIByCategory (retrieved Jan 8, 2023)
// Full source: https://github.com/libsdl-org/SDL/tree/SDL2 (SDL2 branch)
// ASCII art generated by: https://patorjk.com/software/taag/#p=display&f=ANSI%20Shadow&t=Shared%20Object%20Support (with modified 'S' for read


// ██████╗  █████╗  ██████╗██╗ ██████╗ ██████╗
// ██╔══██╗██╔══██╗██╔════╝██║██╔════╝██╔════╝
// ██████╔╝███████║███████╗██║██║     ███████╗
// ██╔══██╗██╔══██║╚════██║██║██║     ╚════██║
// ██████╔╝██║  ██║██████╔╝██║╚██████╗██████╔╝
// ╚═════╝ ╚═╝  ╚═╝╚═════╝ ╚═╝ ╚═════╝╚═════╝

//|-----------------------------------------------------------------------------
//| Init & Quit (SDL.h)
//|-----------------------------------------------------------------------------
int     SDL_Init           (Uint32 flags);  // Initialize the SDL library.
int     SDL_InitSubSystem  (Uint32 flags);  // Compatibility function to initialize the SDL library.
void    SDL_QuitSubSystem  (Uint32 flags);  // Shut down specific SDL subsystems.
Uint32  SDL_WasInit        (Uint32 flags);  // Get a mask of the specified subsystems which are currently initialized.
void    SDL_Quit           (void);          // Clean up all initialized subsystems.

//|-----------------------------------------------------------------------------
//| Config Variables (SDL_hints.h)
//|-----------------------------------------------------------------------------
SDL_bool      SDL_SetHintWithPriority  (const char *name, const char *value, SDL_HintPriority priority);  // Set a hint with a specific priorit
SDL_bool      SDL_SetHint              (const char *name, const char *value);                             // Set a hint with normal priority.
SDL_bool      SDL_ResetHint            (const char *name);                                                // Reset a hint to the default value.
void          SDL_ResetHints           (void);                                                            // Reset all hints to the default val
const char *  SDL_GetHint              (const char *name);                                                // Get the value of a hint.
SDL_bool      SDL_GetHintBoolean       (const char *name, SDL_bool default_value);                        // Get the boolean value of a hint va
void          SDL_AddHintCallback      (const char *name, SDL_HintCallback callback, void *userdata);      // Add a function to watch a particul
void          SDL_DelHintCallback      (const char *name, SDL_HintCallback callback, void *userdata);      // Remove a function watching a parti
void          SDL_ClearHints           (void);                                                            // Clear all hints.

//|-----------------------------------------------------------------------------
//| Error Handling (SDL_error.h)
//|-----------------------------------------------------------------------------
int           SDL_SetError     (const char *fmt, ...);      // Set the SDL error message for the current thread.
const char *  SDL_GetError     (void);                      // Retrieve a message about the last error that occurred on the current thread.
char *        SDL_GetErrorMsg  (char *errstr, int maxlen);  // Get the last error message that was set for the current thread.
void          SDL_ClearError   (void);                      // Clear any previous error message for this thread.

//|-----------------------------------------------------------------------------
//| Logging (SDL_log.h)
//|-----------------------------------------------------------------------------
void             SDL_LogSetAllPriority     (SDL_LogPriority priority);                                             // Set the priority of all l
void             SDL_LogSetPriority        (int category, SDL_LogPriority priority);                               // Set the priority of a par
SDL_LogPriority  SDL_LogGetPriority        (int category);                                                         // Get the priority of a par
void             SDL_LogResetPriorities    (void);                                                                 // Reset all priorities to d
void             SDL_Log                   (const char *fmt, ...);                                                 // Log a message with SDL_LO
void             SDL_LogVerbose            (int category, const char *fmt, ...);                                   // Log a message with SDL_LO
void             SDL_LogDebug              (int category, const char *fmt, ...);                                   // Log a message with SDL_LO
void             SDL_LogInfo               (int category, const char *fmt, ...);                                   // Log a message with SDL_LO
void             SDL_LogWarn               (int category, const char *fmt, ...);                                   // Log a message with SDL_LO
void             SDL_LogError              (int category, const char *fmt, ...);                                   // Log a message with SDL_LO
void             SDL_LogCritical           (int category, const char *fmt, ...);                                   // Log a message with SDL_LO
void             SDL_LogMessage            (int category, SDL_LogPriority priority, const char *fmt, ...);         // Log a message with the sp
void             SDL_LogMessageV           (int category, SDL_LogPriority priority, const char *fmt, va_list ap);  // Log a message with the sp
void             SDL_LogGetOutputFunction  (SDL_LogOutputFunction *callback, void **userdata);                     // Get the current log outpu
void             SDL_LogSetOutputFunction  (SDL_LogOutputFunction callback, void *userdata);                       // Replace the default log o

//|-----------------------------------------------------------------------------
//| Assertions (SDL_assert.h)
//|-----------------------------------------------------------------------------
void                    SDL_SetAssertionHandler         (SDL_AssertionHandler handler, void *userdata);  // Set an application-defined assertio
SDL_AssertionHandler    SDL_GetDefaultAssertionHandler  (void);                                          // Get the default assertion handler.
SDL_AssertionHandler    SDL_GetAssertionHandler         (void **puserdata);                              // Get the current assertion handler.
const SDL_AssertData *  SDL_GetAssertionReport          (void);                                          // Get a list of all assertion failure
void                    SDL_ResetAssertionReport        (void);                                          // Clear the list of all assertion fai

//|-----------------------------------------------------------------------------
//| SDL Version (SDL_version.h, SDL_revision.h)
//|-----------------------------------------------------------------------------
void          SDL_GetVersion   (SDL_version *ver);  // Get the version of SDL that is linked against your program.
const char *  SDL_GetRevision  (void);              // Get the code revision of SDL that is linked against your program.


// ██╗    ██╗██╗███╗   ██╗██████╗  ██████╗ ██╗    ██╗ ██████╗
// ██║    ██║██║████╗  ██║██╔══██╗██╔═══██╗██║    ██║██╔════╝
// ██║ █╗ ██║██║██╔██╗ ██║██║  ██║██║   ██║██║ █╗ ██║███████╗
// ██║███╗██║██║██║╚██╗██║██║  ██║██║   ██║██║███╗██║╚════██║
// ╚███╔███╔╝██║██║ ╚████║██████╔╝╚██████╔╝╚███╔███╔╝██████╔╝
//  ╚══╝╚══╝ ╚═╝╚═╝  ╚═══╝╚═════╝  ╚═════╝  ╚══╝╚══╝ ╚═════╝

//|-----------------------------------------------------------------------------
//| Displays and Windows (SDL_video.h)
//|-----------------------------------------------------------------------------
int                     SDL_GetNumVideoDrivers        (void);                                                                           // Get
const char *            SDL_GetVideoDriver            (int index);                                                                      // Get
int                     SDL_VideoInit                 (const char *driver_name);                                                        // Ini
void                    SDL_VideoQuit                 (void);                                                                           // Shu
const char *            SDL_GetCurrentVideoDriver     (void);                                                                           // Get
```

```c
int                     SDL_GetNumVideoDisplays      (void);                                                                    // Get
const char *            SDL_GetDisplayName           (int displayIndex);                                                        // Get
int                     SDL_GetDisplayBounds         (int displayIndex, SDL_Rect *rect);                                        // Get
int                     SDL_GetDisplayUsableBounds   (int displayIndex, SDL_Rect *rect);                                        // Get
int                     SDL_GetDisplayDPI            (int displayIndex, float *ddpi, float *hdpi, float *vdpi);                 // Get
SDL_DisplayOrientation  SDL_GetDisplayOrientation    (int displayIndex);                                                        // Get
int                     SDL_GetNumDisplayModes       (int displayIndex);                                                        // Get
int                     SDL_GetDisplayMode           (int displayIndex, int modeIndex, SDL_DisplayMode *mode);                  // Get
int                     SDL_GetDesktopDisplayMode    (int displayIndex, SDL_DisplayMode *mode);                                 // Get
int                     SDL_GetCurrentDisplayMode    (int displayIndex, SDL_DisplayMode *mode);                                 // Get
SDL_DisplayMode *       SDL_GetClosestDisplayMode    (int displayIndex, const SDL_DisplayMode *mode, SDL_DisplayMode *closest); // Get
int                     SDL_GetPointDisplayIndex     (const SDL_Point *point);                                                  // Get
int                     SDL_GetRectDisplayIndex      (const SDL_Rect *rect);                                                    // Get
int                     SDL_GetWindowDisplayIndex    (SDL_Window *window);                                                      // Get
int                     SDL_SetWindowDisplayMode     (SDL_Window *window, const SDL_DisplayMode *mode);                         // Set
int                     SDL_GetWindowDisplayMode     (SDL_Window *window, SDL_DisplayMode *mode);                               // Que
void*                   SDL_GetWindowICCProfile      (SDL_Window *window, size_t *size);                                        // Get
Uint32                  SDL_GetWindowPixelFormat     (SDL_Window *window);                                                      // Get
SDL_Window *            SDL_CreateWindow             (const char *title, int x, int y, int w, int h, Uint32 flags);             // Cre
SDL_Window *            SDL_CreateWindowFrom         (const void *data);                                                        // Cre
Uint32                  SDL_GetWindowID              (SDL_Window *window);                                                      // Get
SDL_Window *            SDL_GetWindowFromID          (Uint32 id);                                                               // Get
Uint32                  SDL_GetWindowFlags           (SDL_Window *window);                                                      // Get
void                    SDL_SetWindowTitle           (SDL_Window *window, const char *title);                                   // Set
const char *            SDL_GetWindowTitle           (SDL_Window *window);                                                      // Get
void                    SDL_SetWindowIcon            (SDL_Window *window, SDL_Surface *icon);                                   // Set
void*                   SDL_SetWindowData            (SDL_Window *window, const char *name, void *userdata);                    // Ass
void *                  SDL_GetWindowData            (SDL_Window *window, const char *name);                                    // Ret
void                    SDL_SetWindowPosition        (SDL_Window *window, int x, int y);                                        // Set
void                    SDL_GetWindowPosition        (SDL_Window *window, int *x, int *y);                                      // Get
void                    SDL_SetWindowSize            (SDL_Window *window, int w, int h);                                        // Set
void                    SDL_GetWindowSize            (SDL_Window *window, int *w, int *h);                                      // Get
int                     SDL_GetWindowBordersSize     (SDL_Window *window, int *top, int *left, int *bottom, int *right);        // Get
void                    SDL_GetWindowSizeInPixels    (SDL_Window *window, int *w, int *h);                                      // Get
void                    SDL_SetWindowMinimumSize     (SDL_Window *window, int min_w, int min_h);                                // Set
void                    SDL_GetWindowMinimumSize     (SDL_Window *window, int *w, int *h);                                      // Get
void                    SDL_SetWindowMaximumSize     (SDL_Window *window, int max_w, int max_h);                                // Set
void                    SDL_GetWindowMaximumSize     (SDL_Window *window, int *w, int *h);                                      // Get
void                    SDL_SetWindowBordered        (SDL_Window *window, SDL_bool bordered);                                   // Set
void                    SDL_SetWindowResizable       (SDL_Window *window, SDL_bool resizable);                                  // Set
void                    SDL_SetWindowAlwaysOnTop     (SDL_Window *window, SDL_bool on_top);                                     // Set
void                    SDL_ShowWindow               (SDL_Window *window);                                                      // Sho
void                    SDL_HideWindow               (SDL_Window *window);                                                      // Hid
void                    SDL_RaiseWindow              (SDL_Window *window);                                                      // Rai
void                    SDL_MaximizeWindow           (SDL_Window *window);                                                      // Mak
void                    SDL_MinimizeWindow           (SDL_Window *window);                                                      // Min
void                    SDL_RestoreWindow            (SDL_Window *window);                                                      // Res
int                     SDL_SetWindowFullscreen      (SDL_Window *window, Uint32 flags);                                        // Set
SDL_Surface *           SDL_GetWindowSurface         (SDL_Window *window);                                                      // Get
int                     SDL_UpdateWindowSurface      (SDL_Window *window);                                                      // Cop
int                     SDL_UpdateWindowSurfaceRects (SDL_Window *window, const SDL_Rect *rects, int numrects);                 // Cop
void                    SDL_SetWindowGrab            (SDL_Window *window, SDL_bool grabbed);                                    // Set
void                    SDL_SetWindowKeyboardGrab    (SDL_Window *window, SDL_bool grabbed);                                    // Set
void                    SDL_SetWindowMouseGrab       (SDL_Window *window, SDL_bool grabbed);                                    // Set
SDL_bool                SDL_GetWindowGrab            (SDL_Window *window);                                                      // Get
SDL_bool                SDL_GetWindowKeyboardGrab    (SDL_Window *window);                                                      // Get
SDL_bool                SDL_GetWindowMouseGrab       (SDL_Window *window);                                                      // Get
SDL_Window *            SDL_GetGrabbedWindow         (void);                                                                    // Get
int                     SDL_SetWindowMouseRect       (SDL_Window *window, const SDL_Rect *rect);                                // Con
const SDL_Rect *        SDL_GetWindowMouseRect       (SDL_Window *window);                                                      // Get
int                     SDL_SetWindowBrightness      (SDL_Window *window, float brightness);                                    // Set
float                   SDL_GetWindowBrightness      (SDL_Window *window);                                                      // Get
int                     SDL_SetWindowOpacity         (SDL_Window *window, float opacity);                                       // Set
int                     SDL_GetWindowOpacity         (SDL_Window *window, float *out_opacity);                                  // Get
int                     SDL_SetWindowModalFor        (SDL_Window *modal_window, SDL_Window *parent_window);                     // Set
int                     SDL_SetWindowInputFocus      (SDL_Window *window);                                                      // Exp
int                     SDL_SetWindowGammaRamp       (SDL_Window *window, const Uint16 *red, const Uint16 *green, const Uint16 *blue);  // Set
int                     SDL_GetWindowGammaRamp       (SDL_Window *window, Uint16 *red, Uint16 *green, Uint16 *blue);            // Get
int                     SDL_SetWindowHitTest         (SDL_Window *window, SDL_HitTest callback, void *callback_data);           // Pro
int                     SDL_FlashWindow              (SDL_Window *window, SDL_FlashOperation operation);                        // Req
void                    SDL_DestroyWindow            (SDL_Window *window);                                                      // Des
SDL_bool                SDL_IsScreenSaverEnabled     (void);                                                                    // Che
void                    SDL_EnableScreenSaver        (void);                                                                    // All
void                    SDL_DisableScreenSaver       (void);                                                                    // Pre

//|--------------------------------------------------------------------------------
//| OpenGL support functions (SDL_video.h)
//|--------------------------------------------------------------------------------
int            SDL_GL_LoadLibrary        (const char *path);                            // Dynamically load an OpenGL library.
void *         SDL_GL_GetProcAddress     (const char *proc);                            // Get an OpenGL function by name.
void           SDL_GL_UnloadLibrary      (void);                                        // Unload the OpenGL library previously loaded by SDL_G
SDL_bool       SDL_GL_ExtensionSupported (const char *extension);                       // Check if an OpenGL extension is supported for the cu
void           SDL_GL_ResetAttributes    (void);                                        // Reset all previously set OpenGL context attributes t
int            SDL_GL_SetAttribute       (SDL_GLattr attr, int value);                  // Set an OpenGL window attribute before window creatio
int            SDL_GL_GetAttribute       (SDL_GLattr attr, int *value);                 // Get the actual value for an attribute from the curre
SDL_GLContext  SDL_GL_CreateContext      (SDL_Window *window);                          // Create an OpenGL context for an OpenGL window, and m
int            SDL_GL_MakeCurrent        (SDL_Window *window, SDL_GLContext context);   // Set up an OpenGL context for rendering into an OpenG
SDL_Window*    SDL_GL_GetCurrentWindow   (void);                                        // Get the currently active OpenGL window.
SDL_GLContext  SDL_GL_GetCurrentContext  (void);                                        // Get the currently active OpenGL context.
void           SDL_GL_GetDrawableSize    (SDL_Window *window, int *w, int *h);          // Get the size of a window's underlying drawable in pi
int            SDL_GL_SetSwapInterval    (int interval);                                // Set the swap interval for the current OpenGL context
int            SDL_GL_GetSwapInterval    (void);                                        // Get the swap interval for the current OpenGL context
void           SDL_GL_SwapWindow         (SDL_Window *window);                          // Update a window with OpenGL rendering.
void           SDL_GL_DeleteContext      (SDL_GLContext context);                       // Delete an OpenGL context.

//|--------------------------------------------------------------------------------
//| 2D Accelerated Rendering (SDL_render.h)
//|--------------------------------------------------------------------------------
int             SDL_GetNumRenderDrivers           (void);                                                                         // Ge
int             SDL_GetRenderDriverInfo           (int index, SDL_RendererInfo *info);                                            // Ge
int             SDL_CreateWindowAndRenderer       (int width, int height, Uint32 window_flags, SDL_Window **window,
                                                  SDL_Renderer **renderer);                                                       // Cr
SDL_Renderer *  SDL_CreateRenderer                (SDL_Window *window, int index, Uint32 flags);                                  // Cr
SDL_Renderer *  SDL_CreateSoftwareRenderer        (SDL_Surface *surface);                                                         // Cr
SDL_Renderer *  SDL_GetRenderer                   (SDL_Window *window);                                                           // Ge
SDL_Window *    SDL_RenderGetWindow               (SDL_Renderer *renderer);                                                       // Ge
int             SDL_GetRendererInfo               (SDL_Renderer *renderer, SDL_RendererInfo *info);                               // Ge
int             SDL_GetRendererOutputSize         (SDL_Renderer *renderer, int *w, int *h);                                       // Ge
SDL_Texture *   SDL_CreateTexture                 (SDL_Renderer *renderer, Uint32 format, int access, int w, int h);              // Cr
SDL_Texture *   SDL_CreateTextureFromSurface      (SDL_Renderer *renderer, SDL_Surface *surface);                                 // Cr
int             SDL_QueryTexture                  (SDL_Texture *texture, Uint32 *format, int *access, int *w, int *h);            // Qu
int             SDL_SetTextureColorMod            (SDL_Texture *texture, Uint8 r, Uint8 g, Uint8 b);                              // Se
int             SDL_GetTextureColorMod            (SDL_Texture *texture, Uint8 *r, Uint8 *g, Uint8 *b);                           // Ge
int             SDL_SetTextureAlphaMod            (SDL_Texture *texture, Uint8 alpha);                                            // Se
int             SDL_GetTextureAlphaMod            (SDL_Texture *texture, Uint8 *alpha);                                           // Ge
int             SDL_SetTextureBlendMode           (SDL_Texture *texture, SDL_BlendMode blendMode);                                // Se
int             SDL_GetTextureBlendMode           (SDL_Texture *texture, SDL_BlendMode *blendMode);                               // Ge
int             SDL_SetTextureScaleMode           (SDL_Texture *texture, SDL_ScaleMode scaleMode);                                // Se
int             SDL_GetTextureScaleMode           (SDL_Texture *texture, SDL_ScaleMode *scaleMode);                               // Ge
int             SDL_SetTextureUserData            (SDL_Texture *texture, void *userdata);                                         // As
void *          SDL_GetTextureUserData            (SDL_Texture *texture);                                                         // Ge
int             SDL_UpdateTexture                 (SDL_Texture *texture, const SDL_Rect *rect, const void *pixels, int pitch);    // Up
int             SDL_UpdateYUVTexture              (SDL_Texture *texture, const SDL_Rect *rect, const Uint8 *Yplane, int Ypitch,
                                                  const Uint8 *Uplane, int Upitch, const Uint8 *Vplane, int Vpitch);              // Up
int             SDL_UpdateNVTexture               (SDL_Texture *texture, const SDL_Rect *rect, const Uint8 *Yplane, int Ypitch,
                                                  const Uint8 *UVplane, int UVpitch);                                             // Up
int             SDL_LockTexture                   (SDL_Texture *texture, const SDL_Rect *rect, void **pixels, int *pitch);        // Lo
int             SDL_LockTextureToSurface          (SDL_Texture *texture, const SDL_Rect *rect, SDL_Surface **surface);                    // Lo
void            SDL_UnlockTexture                 (SDL_Texture *texture);                                                                 // Un
SDL_bool        SDL_RenderTargetSupported         (SDL_Renderer *renderer);                                                               // De
int             SDL_SetRenderTarget               (SDL_Renderer *renderer, SDL_Texture *texture);                                         // Se
SDL_Texture *   SDL_GetRenderTarget               (SDL_Renderer *renderer);                                                               // Ge
int             SDL_RenderSetLogicalSize          (SDL_Renderer *renderer, int w, int h);                                                 // Se
void            SDL_RenderGetLogicalSize          (SDL_Renderer *renderer, int *w, int *h);                                               // Ge
int             SDL_RenderSetIntegerScale         (SDL_Renderer *renderer, SDL_bool enable);                                              // Se
SDL_bool        SDL_RenderGetIntegerScale         (SDL_Renderer *renderer);                                                               // Ge
int             SDL_RenderSetViewport             (SDL_Renderer *renderer, const SDL_Rect *rect);                                         // Se
void            SDL_RenderGetViewport             (SDL_Renderer *renderer, SDL_Rect *rect);                                               // Ge
int             SDL_RenderSetClipRect             (SDL_Renderer *renderer, const SDL_Rect *rect);                                         // Se
void            SDL_RenderGetClipRect             (SDL_Renderer *renderer, SDL_Rect *rect);                                               // Ge
SDL_bool        SDL_RenderIsClipEnabled           (SDL_Renderer *renderer);                                                               // Ge
int             SDL_RenderSetScale                (SDL_Renderer *renderer, float scaleX, float scaleY);                                   // Se
void            SDL_RenderGetScale                (SDL_Renderer *renderer, float *scaleX, float *scaleY);                                 // Ge
void            SDL_RenderWindowToLogical         (SDL_Renderer *renderer, int windowX, int windowY, float *logicalX, float *logicalY);   // Ge
void            SDL_RenderLogicalToWindow         (SDL_Renderer *renderer, float logicalX, float logicalY, int *windowX, int *windowY);   // Ge
int             SDL_SetRenderDrawColor            (SDL_Renderer *renderer, Uint8 r, Uint8 g, Uint8 b, Uint8 a);                           // Se
int             SDL_GetRenderDrawColor            (SDL_Renderer *renderer, Uint8 *r, Uint8 *g, Uint8 *b, Uint8 *a);                       // Ge
int             SDL_SetRenderDrawBlendMode        (SDL_Renderer *renderer, SDL_BlendMode blendMode);                                      // Se
int             SDL_GetRenderDrawBlendMode        (SDL_Renderer *renderer, SDL_BlendMode *blendMode);                                     // Ge
int             SDL_RenderClear                   (SDL_Renderer *renderer);                                                               // Cl
int             SDL_RenderDrawPoint               (SDL_Renderer *renderer, int x, int y);                                                 // Dr
int             SDL_RenderDrawPoints              (SDL_Renderer *renderer, const SDL_Point *points, int count);                           // Dr
int             SDL_RenderDrawLine                (SDL_Renderer *renderer, int x1, int y1, int x2, int y2);                               // Dr
int             SDL_RenderDrawLines               (SDL_Renderer *renderer, const SDL_Point *points, int count);                           // Dr
int             SDL_RenderDrawRect                (SDL_Renderer *renderer, const SDL_Rect *rect);                                         // Dr
int             SDL_RenderDrawRects               (SDL_Renderer *renderer, const SDL_Rect *rects, int count);                             // Dr
int             SDL_RenderFillRect                (SDL_Renderer *renderer, const SDL_Rect *rect);                                         // Fi
int             SDL_RenderFillRects               (SDL_Renderer *renderer, const SDL_Rect *rects, int count);                             // Fi
int             SDL_RenderCopy                    (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Rect *srcrect,
                                                  const SDL_Rect *dstrect);                                                               // Co
int             SDL_RenderCopyEx                  (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Rect *srcrect,
                                                  const SDL_Rect *dstrect, const double angle, const SDL_Point *center,
                                                  const SDL_RendererFlip flip);                                                           // Co
int             SDL_RenderDrawPointF              (SDL_Renderer *renderer, float x, float y);                                             // Dr
int             SDL_RenderDrawPointsF             (SDL_Renderer *renderer, const SDL_FPoint *points, int count);                          // Dr
int             SDL_RenderDrawLineF               (SDL_Renderer *renderer, float x1, float y1, float x2, float y2);                       // Dr
int             SDL_RenderDrawLinesF              (SDL_Renderer *renderer, const SDL_FPoint *points, int count);                          // Dr
int             SDL_RenderDrawRectF               (SDL_Renderer *renderer, const SDL_FRect *rect);                                        // Dr
int             SDL_RenderDrawRectsF              (SDL_Renderer *renderer, const SDL_FRect *rects, int count);                            // Dr
int             SDL_RenderFillRectF               (SDL_Renderer *renderer, const SDL_FRect *rect);                                        // Fi
int             SDL_RenderFillRectsF              (SDL_Renderer *renderer, const SDL_FRect *rects, int count);                            // Fi
int             SDL_RenderCopyF                   (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Rect *srcrect,
                                                  const SDL_FRect *dstrect);                                                              // Co
int             SDL_RenderCopyExF                 (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Rect *srcrect,
                                                  const SDL_FRect *dstrect, const double angle, const SDL_FPoint *center,
                                                  const SDL_RendererFlip flip);                                                           // Co
int             SDL_RenderGeometry                (SDL_Renderer *renderer, SDL_Texture *texture, const SDL_Vertex *vertices,
                                                  int num_vertices, const int *indices, int num_indices);                                 // Re
int             SDL_RenderGeometryRaw             (SDL_Renderer *renderer, SDL_Texture *texture, const float *xy, int xy_stride,
                                                  const SDL_Color *color, int color_stride, const float *uv, int uv_stride,
                                                  int num_vertices, const void *indices, int num_indices, int size_indices);              // Re
int             SDL_RenderReadPixels              (SDL_Renderer *renderer, const SDL_Rect *rect, Uint32 format, void *pixels, int pitch); // Re
void            SDL_RenderPresent                 (SDL_Renderer *renderer);                                                               // Up
void            SDL_DestroyTexture                (SDL_Texture *texture);                                                                 // De
void            SDL_DestroyRenderer               (SDL_Renderer *renderer);                                                               // De
int             SDL_RenderFlush                   (SDL_Renderer *renderer);                                                               // Fo
int             SDL_GL_BindTexture                (SDL_Texture *texture, float *texw, float *texh);                                       // Bi
int             SDL_GL_UnbindTexture              (SDL_Texture *texture);                                                                 // Un
void *          SDL_RenderGetMetalLayer           (SDL_Renderer *renderer);                                                               // Ge
void *          SDL_RenderGetMetalCommandEncoder  (SDL_Renderer *renderer);                                                               // Ge
int             SDL_RenderSetVSync                (SDL_Renderer *renderer, int vsync);                                                    // To


//|----------------------------------------------------------------------------
//| Pixel Formats and Conversion Routines (SDL_pixels.h)
//|----------------------------------------------------------------------------
const char*        SDL_GetPixelFormatName      (Uint32 format);                                                                        // Get t
SDL_bool           SDL_PixelFormatEnumToMasks  (Uint32 format, int *bpp, Uint32 *Rmask, Uint32 *Gmask, Uint32 *Bmask, Uint32 *Amask);  // Conve
Uint32             SDL_MasksToPixelFormatEnum  (int bpp, Uint32 Rmask, Uint32 Gmask, Uint32 Bmask, Uint32 Amask);                      // Conve
SDL_PixelFormat *  SDL_AllocFormat             (Uint32 pixel_format);                                                                  // Creat
void               SDL_FreeFormat              (SDL_PixelFormat *format);                                                              // Free
SDL_Palette *      SDL_AllocPalette            (int ncolors);                                                                          // Creat
int                SDL_SetPixelFormatPalette   (SDL_PixelFormat *format, SDL_Palette *palette);                                        // Set t
int                SDL_SetPaletteColors        (SDL_Palette *palette, const SDL_Color *colors, int firstcolor, int ncolors);           // Set a
void               SDL_FreePalette             (SDL_Palette *palette);                                                                 // Free
Uint32             SDL_MapRGB                  (const SDL_PixelFormat *format, Uint8 r, Uint8 g, Uint8 b);                             // Map a
Uint32             SDL_MapRGBA                 (const SDL_PixelFormat *format, Uint8 r, Uint8 g, Uint8 b, Uint8 a);                    // Map a
void               SDL_GetRGB                  (Uint32 pixel, const SDL_PixelFormat *format, Uint8 *r, Uint8 *g, Uint8 *b);            // Get R
void               SDL_GetRGBA                 (Uint32 pixel, const SDL_PixelFormat *format, Uint8 *r, Uint8 *g, Uint8 *b, Uint8 *a);  // Get R
void               SDL_CalculateGammaRamp      (float gamma, Uint16 *ramp);                                                            // Calcu

//|----------------------------------------------------------------------------
//| Rectangle Functions (SDL_rect.h)
//|----------------------------------------------------------------------------
SDL_bool  SDL_PointInRect           (const SDL_Point *p, const SDL_Rect *r)                                        // Returns true if point res
SDL_bool  SDL_RectEmpty             (const SDL_Rect *r)                                                            // Returns true if the recta
SDL_bool  SDL_RectEquals            (const SDL_Rect *a, const SDL_Rect *b)                                         // Returns true if the two r
SDL_bool  SDL_HasIntersection       (const SDL_Rect *A, const SDL_Rect *B);                                        // Determine whether two rec
SDL_bool  SDL_IntersectRect         (const SDL_Rect *A, const SDL_Rect *B, SDL_Rect *result);                      // Calculate the intersectio
void      SDL_UnionRect             (const SDL_Rect *A, const SDL_Rect *B, SDL_Rect *result);                      // Calculate the union of tw
SDL_bool  SDL_EnclosePoints         (const SDL_Point *points, int count, const SDL_Rect *clip, SDL_Rect *result);  // Calculate a minimal recta
SDL_bool  SDL_IntersectRectAndLine  (const SDL_Rect *rect, int *X1, int *Y1, int *X2, int *Y2);                    // Calculate the intersectio

//|----------------------------------------------------------------------------
//| Sub-pixel Rectangle Functions (SDL_rect.h)
//|----------------------------------------------------------------------------
SDL_bool  SDL_PointInFRect           (const SDL_FPoint *p, const SDL_FRect *r)                                         // Returns true if point
SDL_bool  SDL_FRectEmpty             (const SDL_FRect *r)                                                              // Returns true if the r
SDL_bool  SDL_FRectEqualsEpsilon     (const SDL_FRect *a, const SDL_FRect *b, const float epsilon)                     // Returns true if the t
SDL_bool  SDL_FRectEquals            (const SDL_FRect *a, const SDL_FRect *b)                                          // Returns true if the t
SDL_bool  SDL_HasIntersectionF       (const SDL_FRect *A, const SDL_FRect *B);                                         // Determine whether two
SDL_bool  SDL_IntersectFRect         (const SDL_FRect *A, const SDL_FRect *B, SDL_FRect *result);                      // Calculate the interse
void      SDL_UnionFRect             (const SDL_FRect *A, const SDL_FRect *B, SDL_FRect *result);                      // Calculate the union o
SDL_bool  SDL_EncloseFPoints         (const SDL_FPoint *points, int count, const SDL_FRect *clip, SDL_FRect *result);  // Calculate a minimal r
SDL_bool  SDL_IntersectFRectAndLine  (const SDL_FRect *rect, float *X1, float *Y1, float *X2, float *Y2);              // Calculate the interse

//|----------------------------------------------------------------------------
//| Surface Creation and Simple Drawing (SDL_surface.h)
//|----------------------------------------------------------------------------
SDL_Surface *           SDL_CreateRGBSurface                   (Uint32 flags, int width, int height, int depth, Uint32 Rmask, Uint32 Gmask,
                                                               Uint32 Bmask, Uint32 Amask);
SDL_Surface *           SDL_CreateRGBSurfaceWithFormat         (Uint32 flags, int width, int height, int depth, Uint32 format);
SDL_Surface *           SDL_CreateRGBSurfaceFrom               (void *pixels, int width, int height, int depth, int pitch, Uint32 Rmask, Uint32
                                                               Uint32 Bmask, Uint32 Amask);
SDL_Surface *           SDL_CreateRGBSurfaceWithFormatFrom     (void *pixels, int width, int height, int depth, int pitch, Uint32 format);
void                    SDL_FreeSurface                        (SDL_Surface *surface);
int                     SDL_SetSurfacePalette                  (SDL_Surface *surface, SDL_Palette *palette);
int                     SDL_LockSurface                        (SDL_Surface *surface);
void                    SDL_UnlockSurface                      (SDL_Surface *surface);
SDL_Surface *           SDL_LoadBMP_RW                         (SDL_RWops *src, int freesrc);
int                     SDL_SaveBMP_RW                         (SDL_Surface *surface, SDL_RWops *dst, int freedst);
int                     SDL_SetSurfaceRLE                      (SDL_Surface *surface, int flag);
SDL_bool                SDL_HasSurfaceRLE                      (SDL_Surface *surface);
int                     SDL_SetColorKey                        (SDL_Surface *surface, int flag, Uint32 key);
```

```
SDL_bool              SDL_HasColorKey                      (SDL_Surface *surface);
int                   SDL_GetColorKey                      (SDL_Surface *surface, Uint32 *key);
int                   SDL_SetSurfaceColorMod               (SDL_Surface *surface, Uint8 r, Uint8 g, Uint8 b);
int                   SDL_GetSurfaceColorMod               (SDL_Surface *surface, Uint8 *r, Uint8 *g, Uint8 *b);
int                   SDL_SetSurfaceAlphaMod               (SDL_Surface *surface, Uint8 alpha);
int                   SDL_GetSurfaceAlphaMod               (SDL_Surface *surface, Uint8 *alpha);
int                   SDL_SetSurfaceBlendMode              (SDL_Surface *surface, SDL_BlendMode blendMode);
int                   SDL_GetSurfaceBlendMode              (SDL_Surface *surface, SDL_BlendMode *blendMode);
SDL_bool              SDL_SetClipRect                      (SDL_Surface *surface, const SDL_Rect *rect);
void                  SDL_GetClipRect                      (SDL_Surface *surface, SDL_Rect *rect);
SDL_Surface *         SDL_DuplicateSurface                 (SDL_Surface *surface);
SDL_Surface *         SDL_ConvertSurface                   (SDL_Surface *src, const SDL_PixelFormat *fmt, Uint32 flags);
SDL_Surface *         SDL_ConvertSurfaceFormat             (SDL_Surface *src, Uint32 pixel_format, Uint32 flags);
int                   SDL_ConvertPixels                    (int width, int height, Uint32 src_format, const void *src, int src_pitch,
                                                           Uint32 dst_format, void *dst, int dst_pitch);
int                   SDL_PremultiplyAlpha                 (int width, int height, Uint32 src_format, const void *src, int src_pitch,
                                                           Uint32 dst_format, void *dst, int dst_pitch);
int                   SDL_FillRect                         (SDL_Surface *dst, const SDL_Rect *rect, Uint32 color);
int                   SDL_FillRects                        (SDL_Surface *dst, const SDL_Rect *rects, int count, Uint32 color);
int                   SDL_UpperBlit                        (SDL_Surface *src, const SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect)
int                   SDL_LowerBlit                        (SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
int                   SDL_SoftStretch                      (SDL_Surface *src, const SDL_Rect *srcrect, SDL_Surface *dst, const SDL_Rect *ds
int                   SDL_SoftStretchLinear                (SDL_Surface *src, const SDL_Rect *srcrect, SDL_Surface *dst, const SDL_Rect *ds
int                   SDL_UpperBlitScaled                  (SDL_Surface *src, const SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect)
int                   SDL_LowerBlitScaled                  (SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
void                  SDL_SetYUVConversionMode             (SDL_YUV_CONVERSION_MODE mode);
SDL_YUV_CONVERSION_MODE SDL_GetYUVConversionMode           (void);
SDL_YUV_CONVERSION_MODE SDL_GetYUVConversionModeForResolution  (int width, int height);

//|---------------------------------------------------------------------------
//| Platform-specific Window Management (SDL_syswm.h)
//|---------------------------------------------------------------------------
SDL_bool  SDL_GetWindowWMInfo  (SDL_Window *window, SDL_SysWMinfo *info);  // Get driver-specific information about a window.

//|---------------------------------------------------------------------------
//| Clipboard Handling (SDL_clipboard.h)
//|---------------------------------------------------------------------------
int       SDL_SetClipboardText         (const char *text);  // Put UTF-8 text into the clipboard.
char *    SDL_GetClipboardText         (void);              // Get UTF-8 text from the clipboard, which must be freed with SDL_free().
SDL_bool  SDL_HasClipboardText         (void);              // Query whether the clipboard exists and contains a non-empty text string.
int       SDL_SetPrimarySelectionText  (const char *text);  // Put UTF-8 text into the primary selection.
char *    SDL_GetPrimarySelectionText  (void);              // Get UTF-8 text from the primary selection, which must be freed with SDL_free().
SDL_bool  SDL_HasPrimarySelectionText  (void);              // Query whether the primary selection exists and contains a non-empty text string.

//|---------------------------------------------------------------------------
//| Vulkan Support (SDL_vulkan.h)
//|---------------------------------------------------------------------------
int       SDL_Vulkan_LoadLibrary              (const char *path);                                                 // Dynamically load the Vulka
void *    SDL_Vulkan_GetVkGetInstanceProcAddr  (void);                                                            // Get the address of the `vk
void      SDL_Vulkan_UnloadLibrary            (void);                                                             // Unload the Vulkan library
SDL_bool  SDL_Vulkan_GetInstanceExtensions    (SDL_Window *window, unsigned int *pCount, const char **pNames);    // Get the names of the Vulka
SDL_bool  SDL_Vulkan_CreateSurface            (SDL_Window *window, VkInstance instance, VkSurfaceKHR* surface);   // Create a Vulkan rendering
void      SDL_Vulkan_GetDrawableSize          (SDL_Window *window, int *w, int *h);                               // Get the size of the window
```

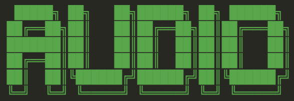


```
//|---------------------------------------------------------------------------
//| Event Handling (SDL_events.h)
//|---------------------------------------------------------------------------
int       SDL_PeepEvents         (SDL_Event *events, int numevents, SDL_eventaction action,
                                  Uint32 minType, Uint32 maxType);           // Check the event queue for messages and optionally return them.
SDL_bool  SDL_HasEvent           (Uint32 type);                              // Check for the existence of a certain event type in the event que
SDL_bool  SDL_HasEvents          (Uint32 minType, Uint32 maxType);           // Check for the existence of certain event types in the event queu
void      SDL_FlushEvent         (Uint32 type);                              // Clear events of a specific type from the event queue.
void      SDL_FlushEvents        (Uint32 minType, Uint32 maxType);           // Clear events of a range of types from the event queue.
int       SDL_PollEvent          (SDL_Event *event);                         // Poll for currently pending events.
int       SDL_WaitEvent          (SDL_Event *event);                         // Wait indefinitely for the next available event.
int       SDL_WaitEventTimeout   (SDL_Event *event, int timeout);            // Wait until the specified timeout (in milliseconds) for the next
int       SDL_PushEvent          (SDL_Event *event);                         // Add an event to the event queue.
void      SDL_SetEventFilter     (SDL_EventFilter filter, void *userdata);   // Set up a filter to process all events before they change interna
SDL_bool  SDL_GetEventFilter     (SDL_EventFilter *filter, void **userdata); // Query the current event filter.
void      SDL_AddEventWatch      (SDL_EventFilter filter, void *userdata);   // Add a callback to be triggered when an event is added to the eve
void      SDL_DelEventWatch      (SDL_EventFilter filter, void *userdata);   // Remove an event watch callback added with SDL_AddEventWatch().
void      SDL_FilterEvents       (SDL_EventFilter filter, void *userdata);   // Run a specific filter function on the current event queue, remov
Uint8     SDL_EventState         (Uint32 type, int state);                   // Set the state of processing events by type.
Uint32    SDL_RegisterEvents     (int numevents);                            // Allocate a set of user-defined events, and return the beginning
```

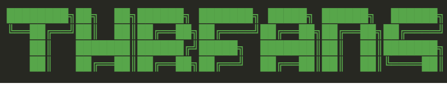


```
//|---------------------------------------------------------------------------
//| Keyboard Support (SDL_keyboard.h)
//|---------------------------------------------------------------------------
const Uint8 *  SDL_GetKeyboardState         (int *numkeys);           // Get a snapshot of the current state of the keyboard.
SDL_Window *   SDL_GetKeyboardFocus         (void);                   // Query the window which currently has keyboard focus.
void           SDL_ResetKeyboard            (void);                   // Clear the state of the keyboard
SDL_Keymod     SDL_GetModState              (void);                   // Get the current key modifier state for the keyboard.
void           SDL_SetModState              (SDL_Keymod modstate);    // Set the current key modifier state for the keyboard.
SDL_Keycode    SDL_GetKeyFromScancode       (SDL_Scancode scancode);  // Get the key code corresponding to the given scancode according to the
SDL_Scancode   SDL_GetScancodeFromKey       (SDL_Keycode key);        // Get the scancode corresponding to the given key code according to the
const char *   SDL_GetScancodeName          (SDL_Scancode scancode);  // Get a human-readable name for a scancode.
SDL_Scancode   SDL_GetScancodeFromName      (const char *name);       // Get a scancode from a human-readable name.
const char *   SDL_GetKeyName               (SDL_Keycode key);        // Get a human-readable name for a key.
SDL_Keycode    SDL_GetKeyFromName           (const char *name);       // Get a key code from a human-readable name.
void           SDL_StartTextInput           (void);                   // Start accepting Unicode text input events.
SDL_bool       SDL_IsTextInputActive        (void);                   // Check whether or not Unicode text input events are enabled.
void           SDL_StopTextInput            (void);                   // Stop receiving any text input events.
void           SDL_ClearComposition         (void);                   // Dismiss the composition window/IME without disabling the subsystem.
SDL_bool       SDL_IsTextInputShown         (void);                   // Returns if an IME Composite or Candidate window is currently shown.
void           SDL_SetTextInputRect         (const SDL_Rect *rect);   // Set the rectangle used to type Unicode text inputs.
SDL_bool       SDL_HasScreenKeyboardSupport (void);                   // Check whether the platform has screen keyboard support.
SDL_bool       SDL_IsScreenKeyboardShown    (SDL_Window *window);     // Check whether the screen keyboard is shown for given window.

//|---------------------------------------------------------------------------
//| Mouse Support (SDL_mouse.h)
//|---------------------------------------------------------------------------
SDL_Window *  SDL_GetMouseFocus          (void);                                       // Get the window which currently has mouse focus.
Uint32        SDL_GetMouseState          (int *x, int *y);                             // Retrieve the current state of the mouse.
Uint32        SDL_GetGlobalMouseState    (int *x, int *y);                             // Get the current state of the mouse in relation to th
Uint32        SDL_GetRelativeMouseState  (int *x, int *y);                             // Retrieve the relative state of the mouse.
void          SDL_WarpMouseInWindow      (SDL_Window *window, int x, int y);           // Move the mouse cursor to the given position within t
int           SDL_WarpMouseGlobal        (int x, int y);                               // Move the mouse to the given position in global scree
int           SDL_SetRelativeMouseMode   (SDL_bool enabled);                           // Set relative mouse mode.
int           SDL_CaptureMouse           (SDL_bool enabled);                           // Capture the mouse and to track input outside an SDL
SDL_bool      SDL_GetRelativeMouseMode   (void);                                       // Query whether relative mouse mode is enabled.
SDL_Cursor *  SDL_CreateCursor           (const Uint8 *data, const Uint8 *mask, int w,
                                         int h, int hot_x, int hot_y);                  // Create a cursor using the specified bitmap data and
SDL_Cursor *  SDL_CreateColorCursor      (SDL_Surface *surface, int hot_x, int hot_y);  // Create a color cursor.
SDL_Cursor *  SDL_CreateSystemCursor     (SDL_SystemCursor id);                         // Create a system cursor.
void          SDL_SetCursor              (SDL_Cursor *cursor);                          // Set the active cursor.
SDL_Cursor *  SDL_GetCursor              (void);                                        // Get the active cursor.
SDL_Cursor *  SDL_GetDefaultCursor       (void);                                        // Get the default cursor.
void          SDL_FreeCursor             (SDL_Cursor *cursor);                          // Free a previously-created cursor.
int           SDL_ShowCursor             (int toggle);                                  // Toggle whether or not the cursor is shown.

//|----------------------------------------------------------------------------
//| Joystick Support (SDL_joystick.h)
//|----------------------------------------------------------------------------
void                    SDL_LockJoysticks                    (void);                                                        // Locking for mult
void                    SDL_UnlockJoysticks                  (void);                                                        // Unlocking for mu
int                     SDL_NumJoysticks                     (void);                                                        // Count the number
const char *            SDL_JoystickNameForIndex             (int device_index);                                            // Get the implemen
const char *            SDL_JoystickPathForIndex             (int device_index);                                            // Get the implemen
int                     SDL_JoystickGetDevicePlayerIndex     (int device_index);                                            // Get the player i
SDL_JoystickGUID        SDL_JoystickGetDeviceGUID            (int device_index);                                            // Get the implemen
Uint16                  SDL_JoystickGetDeviceVendor          (int device_index);                                            // Get the USB venc
Uint16                  SDL_JoystickGetDeviceProduct         (int device_index);                                            // Get the USB proc
Uint16                  SDL_JoystickGetDeviceProductVersion  (int device_index);                                            // Get the product
SDL_JoystickType        SDL_JoystickGetDeviceType            (int device_index);                                            // Get the type of
SDL_JoystickID          SDL_JoystickGetDeviceInstanceID      (int device_index);                                            // Get the instance
SDL_Joystick *          SDL_JoystickOpen                     (int device_index);                                            // Open a joystick
SDL_Joystick *          SDL_JoystickFromInstanceID           (SDL_JoystickID instance_id);                                  // Get the SDL_Joys
SDL_Joystick *          SDL_JoystickFromPlayerIndex          (int player_index);                                            // Get the SDL_Joys
int                     SDL_JoystickAttachVirtual            (SDL_JoystickType type, int naxes, int nbuttons, int nhats);   // Attach a new vir
int                     SDL_JoystickAttachVirtualEx          (const SDL_VirtualJoystickDesc *desc);                         // Attach a new vir
int                     SDL_JoystickDetachVirtual            (int device_index);                                            // Detach a virtual
SDL_bool                SDL_JoystickIsVirtual                (int device_index);                                            // Query whether or
int                     SDL_JoystickSetVirtualAxis           (SDL_Joystick *joystick, int axis, Sint16 value);              // Set values on ar
int                     SDL_JoystickSetVirtualButton         (SDL_Joystick *joystick, int button, Uint8 value);             // Set values on ar
int                     SDL_JoystickSetVirtualHat            (SDL_Joystick *joystick, int hat, Uint8 value);                // Set values on ar
const char *            SDL_JoystickName                     (SDL_Joystick *joystick);                                      // Get the implemen
const char *            SDL_JoystickPath                     (SDL_Joystick *joystick);                                      // Get the implemen
int                     SDL_JoystickGetPlayerIndex           (SDL_Joystick *joystick);                                      // Get the player i
void                    SDL_JoystickSetPlayerIndex           (SDL_Joystick *joystick, int player_index);                    // Set the player i
SDL_JoystickGUID        SDL_JoystickGetGUID                  (SDL_Joystick *joystick);                                      // Get the implemen
Uint16                  SDL_JoystickGetVendor                (SDL_Joystick *joystick);                                      // Get the USB venc
Uint16                  SDL_JoystickGetProduct               (SDL_Joystick *joystick);                                      // Get the USB proc
Uint16                  SDL_JoystickGetProductVersion        (SDL_Joystick *joystick);                                      // Get the product
Uint16                  SDL_JoystickGetFirmwareVersion       (SDL_Joystick *joystick);                                      // Get the firmware
const char *            SDL_JoystickGetSerial                (SDL_Joystick *joystick);                                      // Get the serial r
SDL_JoystickType        SDL_JoystickGetType                  (SDL_Joystick *joystick);                                      // Get the type of
void                    SDL_JoystickGetGUIDString            (SDL_JoystickGUID guid, char *pszGUID, int cbGUID);            // Get an ASCII str
SDL_JoystickGUID        SDL_JoystickGetGUIDFromString        (const char *pchGUID);                                         // Convert a GUID s
void                    SDL_GetJoystickGUIDInfo              (SDL_JoystickGUID guid, Uint16 *vendor, Uint16 *product,
                                                             Uint16 *version, Uint16 *crc16);                               // Get the device i
SDL_bool                SDL_JoystickGetAttached              (SDL_Joystick *joystick);                                      // Get the status c
SDL_JoystickID          SDL_JoystickInstanceID               (SDL_Joystick *joystick);                                      // Get the instance
int                     SDL_JoystickNumAxes                  (SDL_Joystick *joystick);                                      // Get the number c
int                     SDL_JoystickNumBalls                 (SDL_Joystick *joystick);                                      // Get the number c
int                     SDL_JoystickNumHats                  (SDL_Joystick *joystick);                                      // Get the number c
int                     SDL_JoystickNumButtons               (SDL_Joystick *joystick);                                      // Get the number c
void                    SDL_JoystickUpdate                   (void);                                                        // Update the curre
int                     SDL_JoystickEventState               (int state);                                                   // Enable/disable j
Sint16                  SDL_JoystickGetAxis                  (SDL_Joystick *joystick, int axis);                            // Get the current
SDL_bool                SDL_JoystickGetAxisInitialState      (SDL_Joystick *joystick, int axis, Sint16 *state);             // Get the initial
Uint8                   SDL_JoystickGetHat                   (SDL_Joystick *joystick, int hat);                             // Get the current
int                     SDL_JoystickGetBall                  (SDL_Joystick *joystick, int ball, int *dx, int *dy);          // Get the ball axi
Uint8                   SDL_JoystickGetButton                (SDL_Joystick *joystick, int button);                          // Get the current
int                     SDL_JoystickRumble                   (SDL_Joystick *joystick, Uint16 low_frequency_rumble,
                                                             Uint16 high_frequency_rumble, Uint32 duration_ms);             // Start a rumble e
int                     SDL_JoystickRumbleTriggers           (SDL_Joystick *joystick, Uint16 left_rumble,
                                                             Uint16 right_rumble, Uint32 duration_ms);                      // Start a rumble e
SDL_bool                SDL_JoystickHasLED                   (SDL_Joystick *joystick);                                      // Query whether a
SDL_bool                SDL_JoystickHasRumble                (SDL_Joystick *joystick);                                      // Query whether a
SDL_bool                SDL_JoystickHasRumbleTriggers        (SDL_Joystick *joystick);                                      // Query whether a
int                     SDL_JoystickSetLED                   (SDL_Joystick *joystick, Uint8 red, Uint8 green, Uint8 blue);  // Update a joystic
int                     SDL_JoystickSendEffect               (SDL_Joystick *joystick, const void *data, int size);          // Send a joystick
void                    SDL_JoystickClose                    (SDL_Joystick *joystick);                                      // Close a joystick
SDL_JoystickPowerLevel  SDL_JoystickCurrentPowerLevel        (SDL_Joystick *joystick);                                      // Get the battery

//|----------------------------------------------------------------------------
//| Game Controller Support (SDL_gamecontroller.h)
//|----------------------------------------------------------------------------
int                           SDL_GameControllerAddMappingsFromRW               (SDL_RWops *rw, int freerw);
int                           SDL_GameControllerAddMapping                      (const char* mappingString);
int                           SDL_GameControllerNumMappings                     (void);
char *                        SDL_GameControllerMappingForIndex                 (int mapping_index);
char *                        SDL_GameControllerMappingForGUID                  (SDL_JoystickGUID guid);
char *                        SDL_GameControllerMapping                         (SDL_GameController *gamecontroller);
SDL_bool                      SDL_IsGameController                              (int joystick_index);
const char *                  SDL_GameControllerNameForIndex                    (int joystick_index);
const char *                  SDL_GameControllerPathForIndex                    (int joystick_index);
SDL_GameControllerType        SDL_GameControllerTypeForIndex                    (int joystick_index);
char *                        SDL_GameControllerMappingForDeviceIndex           (int joystick_index);
SDL_GameController *          SDL_GameControllerOpen                            (int joystick_index);
SDL_GameController *          SDL_GameControllerFromInstanceID                  (SDL_JoystickID joyid);
SDL_GameController *          SDL_GameControllerFromPlayerIndex                 (int player_index);
const char *                  SDL_GameControllerName                            (SDL_GameController *gamecontroller);
const char *                  SDL_GameControllerPath                            (SDL_GameController *gamecontroller);
SDL_GameControllerType        SDL_GameControllerGetType                         (SDL_GameController *gamecontroller);
int                           SDL_GameControllerGetPlayerIndex                  (SDL_GameController *gamecontroller);
void                          SDL_GameControllerSetPlayerIndex                  (SDL_GameController *gamecontroller, int player_index);
Uint16                        SDL_GameControllerGetVendor                       (SDL_GameController *gamecontroller);
Uint16                        SDL_GameControllerGetProduct                      (SDL_GameController *gamecontroller);
Uint16                        SDL_GameControllerGetProductVersion               (SDL_GameController *gamecontroller);
Uint16                        SDL_GameControllerGetFirmwareVersion              (SDL_GameController *gamecontroller);
const char *                  SDL_GameControllerGetSerial                       (SDL_GameController *gamecontroller);
SDL_bool                      SDL_GameControllerGetAttached                     (SDL_GameController *gamecontroller);
SDL_Joystick *                SDL_GameControllerGetJoystick                     (SDL_GameController *gamecontroller);
int                           SDL_GameControllerEventState                      (int state);
void                          SDL_GameControllerUpdate                          (void);
SDL_GameControllerAxis        SDL_GameControllerGetAxisFromString               (const char *str);
const char*                   SDL_GameControllerGetStringForAxis                (SDL_GameControllerAxis axis);
SDL_GameControllerButtonBind  SDL_GameControllerGetBindForAxis                  (SDL_GameController *gamecontroller, SDL_GameControllerAxis axi
SDL_bool                      SDL_GameControllerHasAxis                         (SDL_GameController *gamecontroller, SDL_GameControllerAxis axi
Sint16                        SDL_GameControllerGetAxis                         (SDL_GameController *gamecontroller, SDL_GameControllerAxis axi
SDL_GameControllerButton      SDL_GameControllerGetButtonFromString             (const char *str);
const char*                   SDL_GameControllerGetStringForButton              (SDL_GameControllerButton button);
SDL_GameControllerButtonBind  SDL_GameControllerGetBindForButton                (SDL_GameController *gamecontroller, SDL_GameControllerButton b
SDL_bool                      SDL_GameControllerHasButton                       (SDL_GameController *gamecontroller, SDL_GameControllerButton b
Uint8                         SDL_GameControllerGetButton                       (SDL_GameController *gamecontroller, SDL_GameControllerButton b
int                           SDL_GameControllerGetNumTouchpads                 (SDL_GameController *gamecontroller);
int                           SDL_GameControllerGetNumTouchpadFingers           (SDL_GameController *gamecontroller, int touchpad);
int                           SDL_GameControllerGetTouchpadFinger               (SDL_GameController *gamecontroller, int touchpad, int finger,
                                                                                float *x, float *y, float *pressure);
SDL_bool                      SDL_GameControllerHasSensor                       (SDL_GameController *gamecontroller, SDL_SensorType type);
int                           SDL_GameControllerSetSensorEnabled                (SDL_GameController *gamecontroller, SDL_SensorType type, SDL_b
SDL_bool                      SDL_GameControllerIsSensorEnabled                 (SDL_GameController *gamecontroller, SDL_SensorType type);
float                         SDL_GameControllerGetSensorDataRate               (SDL_GameController *gamecontroller, SDL_SensorType type);
int                           SDL_GameControllerGetSensorData                   (SDL_GameController *gamecontroller, SDL_SensorType type, float
                                                                                int num_values);
int                           SDL_GameControllerGetSensorDataWithTimestamp      (SDL_GameController *gamecontroller, SDL_SensorType type, Uint6
                                                                                float *data, int num_values);
```

```
int                           SDL_GameControllerRumble                          (SDL_GameController *gamecontroller, Uint16 low_frequency_rumbl
                                                                                Uint16 high_frequency_rumble, Uint32 duration_ms);
int                           SDL_GameControllerRumbleTriggers                  (SDL_GameController *gamecontroller, Uint16 left_rumble, Uint16
                                                                                Uint32 duration_ms);
SDL_bool                      SDL_GameControllerHasLED                          (SDL_GameController *gamecontroller);
SDL_bool                      SDL_GameControllerHasRumble                       (SDL_GameController *gamecontroller);
SDL_bool                      SDL_GameControllerHasRumbleTriggers               (SDL_GameController *gamecontroller);
int                           SDL_GameControllerSetLED                          (SDL_GameController *gamecontroller, Uint8 red, Uint8 green, Ui
int                           SDL_GameControllerSendEffect                      (SDL_GameController *gamecontroller, const void *data, int size
void                          SDL_GameControllerClose                           (SDL_GameController *gamecontroller);
const char*                   SDL_GameControllerGetAppleSFSymbolsNameForButton  (SDL_GameController *gamecontroller, SDL_GameControllerButton b
const char*                   SDL_GameControllerGetAppleSFSymbolsNameForAxis    (SDL_GameController *gamecontroller, SDL_GameControllerAxis axi

//|----------------------------------------------------------------------------
//| Sensors (SDL_sensor.h)
//|----------------------------------------------------------------------------
void            SDL_LockSensors                    (void);                                                                       // Lock for multi-th
void            SDL_UnlockSensors                  (void);                                                                       // Unlock for multi-
int             SDL_NumSensors                     (void);                                                                       // Count the number
const char *    SDL_SensorGetDeviceName            (int device_index);                                                           // Get the implement
SDL_SensorType  SDL_SensorGetDeviceType            (int device_index);                                                           // Get the type of a
int             SDL_SensorGetDeviceNonPortableType (int device_index);                                                           // Get the platform
SDL_SensorID    SDL_SensorGetDeviceInstanceID      (int device_index);                                                           // Get the instance
SDL_Sensor *    SDL_SensorOpen                     (int device_index);                                                           // Open a sensor for
SDL_Sensor *    SDL_SensorFromInstanceID           (SDL_SensorID instance_id);                                                   // Return the SDL_Se
const char *    SDL_SensorGetName                  (SDL_Sensor *sensor);                                                         // Get the implement
SDL_SensorType  SDL_SensorGetType                  (SDL_Sensor *sensor);                                                         // Get the type of a
int             SDL_SensorGetNonPortableType       (SDL_Sensor *sensor);                                                         // Get the platform
SDL_SensorID    SDL_SensorGetInstanceID            (SDL_Sensor *sensor);                                                         // Get the instance
int             SDL_SensorGetData                  (SDL_Sensor *sensor, float *data, int num_values);                            // Get the current s
int             SDL_SensorGetDataWithTimestamp     (SDL_Sensor *sensor, Uint64 *timestamp, float *data, int num_values);         // Get the current s
void            SDL_SensorClose                    (SDL_Sensor *sensor);                                                         // Close a sensor pr
void            SDL_SensorUpdate                   (void);                                                                       // Update the curren

//|----------------------------------------------------------------------------
//| Force Feedback Support (SDL_haptic.h)
//|----------------------------------------------------------------------------
int           SDL_NumHaptics              (void);                                                     // Count the number of haptic devices att
const char *  SDL_HapticName              (int device_index);                                         // Get the implementation dependent name
SDL_Haptic *  SDL_HapticOpen              (int device_index);                                         // Open a haptic device for use.
int           SDL_HapticOpened            (int device_index);                                         // Check if the haptic device at the desi
int           SDL_HapticIndex             (SDL_Haptic *haptic);                                       // Get the index of a haptic device.
int           SDL_MouseIsHaptic           (void);                                                     // Query whether or not the current mouse
SDL_Haptic *  SDL_HapticOpenFromMouse     (void);                                                     // Try to open a haptic device from the c
int           SDL_JoystickIsHaptic        (SDL_Joystick *joystick);                                   // Query if a joystick has haptic feature
SDL_Haptic *  SDL_HapticOpenFromJoystick  (SDL_Joystick *joystick);                                   // Open a haptic device for use from a jo
void          SDL_HapticClose             (SDL_Haptic *haptic);                                       // Close a haptic device previously opene
int           SDL_HapticNumEffects        (SDL_Haptic *haptic);                                       // Get the number of effects a haptic dev
int           SDL_HapticNumEffectsPlaying (SDL_Haptic *haptic);                                       // Get the number of effects a haptic dev
unsigned int  SDL_HapticQuery             (SDL_Haptic *haptic);                                       // Get the haptic device's supported feat
int           SDL_HapticNumAxes           (SDL_Haptic *haptic);                                       // Get the number of haptic axes the devi
int           SDL_HapticEffectSupported   (SDL_Haptic *haptic, SDL_HapticEffect *effect);             // Check to see if an effect is supported
int           SDL_HapticNewEffect         (SDL_Haptic *haptic, SDL_HapticEffect *effect);             // Create a new haptic effect on a specif
int           SDL_HapticUpdateEffect      (SDL_Haptic *haptic, int effect, SDL_HapticEffect *data);   // Update the properties of an effect.
int           SDL_HapticRunEffect         (SDL_Haptic *haptic, int effect, Uint32 iterations);        // Run the haptic effect on its associate
int           SDL_HapticStopEffect        (SDL_Haptic *haptic, int effect);                           // Stop the haptic effect on its associat
void          SDL_HapticDestroyEffect     (SDL_Haptic *haptic, int effect);                           // Destroy a haptic effect on the device.
int           SDL_HapticGetEffectStatus   (SDL_Haptic *haptic, int effect);                           // Get the status of the current effect o
int           SDL_HapticSetGain           (SDL_Haptic *haptic, int gain);                             // Set the global gain of the specified h
int           SDL_HapticSetAutocenter     (SDL_Haptic *haptic, int autocenter);                       // Set the global autocenter of the devic
int           SDL_HapticPause             (SDL_Haptic *haptic);                                       // Pause a haptic device.
int           SDL_HapticUnpause           (SDL_Haptic *haptic);                                       // Unpause a haptic device.
int           SDL_HapticStopAll           (SDL_Haptic *haptic);                                       // Stop all the currently playing effects
int           SDL_HapticRumbleSupported   (SDL_Haptic *haptic);                                       // Check whether rumble is supported on a
int           SDL_HapticRumbleInit        (SDL_Haptic *haptic);                                       // Initialize a haptic device for simple
int           SDL_HapticRumblePlay        (SDL_Haptic *haptic, float strength, Uint32 length );       // Run a simple rumble effect on a haptic
int           SDL_HapticRumbleStop        (SDL_Haptic *haptic);                                       // Stop the simple rumble on a haptic dev


//
//
//
//
//
//
```

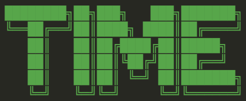


```

//|----------------------------------------------------------------------------
//| Audio Device Management, Playing and Recording (SDL_audio.h)
//|----------------------------------------------------------------------------
int                SDL_GetNumAudioDrivers     (void);                                                                              // Use this fu
const char *       SDL_GetAudioDriver         (int index);                                                                         // Use this fu
int                SDL_AudioInit              (const char *driver_name);                                                           // Use this fu
void               SDL_AudioQuit              (void);                                                                              // Use this fu
const char *       SDL_GetCurrentAudioDriver  (void);                                                                              // Get the nam
int                SDL_OpenAudio              (SDL_AudioSpec *desired, SDL_AudioSpec *obtained);                                   // This functi
int                SDL_GetNumAudioDevices     (int iscapture);                                                                     // Get the num
const char *       SDL_GetAudioDeviceName     (int index, int iscapture);                                                          // Get the hum
int                SDL_GetAudioDeviceSpec     (int index, int iscapture, SDL_AudioSpec *spec);                                     // Get the pre
int                SDL_GetDefaultAudioInfo    (char **name, SDL_AudioSpec *spec, int iscapture);                                   // Get the nam
SDL_AudioDeviceID  SDL_OpenAudioDevice        (const char *device, int iscapture, const SDL_AudioSpec *desired,
                                              SDL_AudioSpec *obtained, int allowed_changes);                                       // Open a spec
SDL_AudioStatus    SDL_GetAudioStatus         (void);                                                                              // This functi
SDL_AudioStatus    SDL_GetAudioDeviceStatus   (SDL_AudioDeviceID dev);                                                             // Use this fu
void               SDL_PauseAudio             (int pause_on);                                                                      // This functi
void               SDL_PauseAudioDevice       (SDL_AudioDeviceID dev, int pause_on);                                               // Use this fu
SDL_AudioSpec *    SDL_LoadWAV_RW             (SDL_RWops *src, int freesrc, SDL_AudioSpec *spec, Uint8 **audio_buf,
                                              Uint32 *audio_len);                                                                  // Load the au
void               SDL_FreeWAV                (Uint8 *audio_buf);                                                                  // Free data p
int                SDL_BuildAudioCVT          (SDL_AudioCVT *cvt, SDL_AudioFormat src_format, Uint8 src_channels, int src_rate,
                                              SDL_AudioFormat dst_format, Uint8 dst_channels, int dst_rate);                       // Initialize
int                SDL_ConvertAudio           (SDL_AudioCVT *cvt);                                                                 // Convert aud
SDL_AudioStream *  SDL_NewAudioStream         (const SDL_AudioFormat src_format, const Uint8 src_channels, const int src_rate,
                                              const SDL_AudioFormat dst_format, const Uint8 dst_channels, const int dst_rate);     // Create a ne
int                SDL_AudioStreamPut         (SDL_AudioStream *stream, const void *buf, int len);                                 // Add data to
int                SDL_AudioStreamGet         (SDL_AudioStream *stream, void *buf, int len);                                       // Get convert
int                SDL_AudioStreamAvailable   (SDL_AudioStream *stream);                                                           // Get the num
int                SDL_AudioStreamFlush       (SDL_AudioStream *stream);                                                           // Tell the st
void               SDL_AudioStreamClear       (SDL_AudioStream *stream);                                                           // Clear any p
void               SDL_FreeAudioStream        (SDL_AudioStream *stream);                                                           // Free an aud
void               SDL_MixAudio               (Uint8 *dst, const Uint8 *src, Uint32 len, int volume);                              // This functi
void               SDL_MixAudioFormat         (Uint8 *dst, const Uint8 *src, SDL_AudioFormat format, Uint32 len, int volume);      // Mix audio d
int                SDL_QueueAudio             (SDL_AudioDeviceID dev, const void *data, Uint32 len);                               // Queue more
Uint32             SDL_DequeueAudio           (SDL_AudioDeviceID dev, void *data, Uint32 len);                                     // Dequeue mor
Uint32             SDL_GetQueuedAudioSize     (SDL_AudioDeviceID dev);                                                             // Get the num
void               SDL_ClearQueuedAudio       (SDL_AudioDeviceID dev);                                                             // Drop any qu
void               SDL_LockAudio              (void);                                                                              // This functi
void               SDL_LockAudioDevice        (SDL_AudioDeviceID dev);                                                             // Use this fu
void               SDL_UnlockAudio            (void);                                                                              // This functi
void               SDL_UnlockAudioDevice      (SDL_AudioDeviceID dev);                                                             // Use this fu
void               SDL_CloseAudio             (void);                                                                              // This functi
void               SDL_CloseAudioDevice       (SDL_AudioDeviceID dev);                                                             // Use this fu


//
//
//
//
```

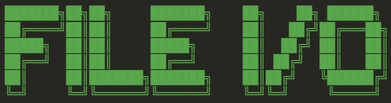


```
//
//
```

```

//|--------------------------------------------------------------------------
//| Thread Management (SDL_thread.h)
//|--------------------------------------------------------------------------
SDL_Thread *  SDL_CreateThread              (SDL_ThreadFunction fn, const char *name, void *data);                      // Create a new th
SDL_Thread *  SDL_CreateThreadWithStackSize (SDL_ThreadFunction fn, const char *name, const size_t stacksize, void *data);  // Create a new th
const char *  SDL_GetThreadName             (SDL_Thread *thread);                                                        // Get the thread
SDL_threadID  SDL_ThreadID                  (void);                                                                      // Get the thread
SDL_threadID  SDL_GetThreadID               (SDL_Thread *thread);                                                        // Get the thread
int           SDL_SetThreadPriority         (SDL_ThreadPriority priority);                                               // Set the priorit
void          SDL_WaitThread                (SDL_Thread *thread, int *status);                                           // Wait for a thre
void          SDL_DetachThread              (SDL_Thread *thread);                                                        // Let a thread cl
SDL_TLSID     SDL_TLSCreate                 (void);                                                                      // Create a piece
void *        SDL_TLSGet                    (SDL_TLSID id);                                                              // Get the current
int           SDL_TLSSet                    (SDL_TLSID id, const void *value, void (*destructor)(void*));                // Set the current
void          SDL_TLSCleanup                (void);                                                                      // Cleanup all TLS

//|--------------------------------------------------------------------------
//| Thread Synchronization Primitives (SDL_mutex.h)
//|--------------------------------------------------------------------------
SDL_mutex *  SDL_CreateMutex       (void);                                        // Create a new mutex.
int          SDL_LockMutex         (SDL_mutex *mutex) SDL_ACQUIRE(mutex);         // Lock the mutex.
int          SDL_TryLockMutex      (SDL_mutex *mutex) SDL_TRY_ACQUIRE(0, mutex);  // Try to lock a mutex without blocking.
int          SDL_UnlockMutex       (SDL_mutex *mutex) SDL_RELEASE(mutex);         // Unlock the mutex.
void         SDL_DestroyMutex      (SDL_mutex *mutex);                            // Destroy a mutex created with SDL_CreateMutex().
SDL_sem *    SDL_CreateSemaphore   (Uint32 initial_value);                        // Create a semaphore.
void         SDL_DestroySemaphore  (SDL_sem *sem);                                // Destroy a semaphore.
int          SDL_SemWait           (SDL_sem *sem);                                // Wait until a semaphore has a positive value and then decr
int          SDL_SemTryWait        (SDL_sem *sem);                                // See if a semaphore has a positive value and decrement it
int          SDL_SemWaitTimeout    (SDL_sem *sem, Uint32 timeout);                // Wait until a semaphore has a positive value and then decr
int          SDL_SemPost           (SDL_sem *sem);                                // Atomically increment a semaphore's value and wake waiting
Uint32       SDL_SemValue          (SDL_sem *sem);                                // Get the current value of a semaphore.
SDL_cond *   SDL_CreateCond        (void);                                        // Create a condition variable.
void         SDL_DestroyCond       (SDL_cond *cond);                              // Destroy a condition variable.
int          SDL_CondSignal        (SDL_cond *cond);                              // Restart one of the threads that are waiting on the condit
int          SDL_CondBroadcast     (SDL_cond *cond);                              // Restart all threads that are waiting on the condition var
int          SDL_CondWait          (SDL_cond *cond, SDL_mutex *mutex);            // Wait until a condition variable is signaled.
int          SDL_CondWaitTimeout   (SDL_cond *cond, SDL_mutex *mutex, Uint32 ms); // Wait until a condition variable is signaled or a certain

//|--------------------------------------------------------------------------
//| Atomic Operations (SDL_atomic.h)
//|--------------------------------------------------------------------------
SDL_bool  SDL_AtomicTryLock         (SDL_SpinLock *lock);                       // Try to lock a spin lock by setting it to a non-zero value.
void      SDL_AtomicLock            (SDL_SpinLock *lock);                       // Lock a spin lock by setting it to a non-zero value.
void      SDL_AtomicUnlock          (SDL_SpinLock *lock);                       // Unlock a spin lock by setting it to 0.
void      SDL_MemoryBarrierAcquire  (void);                                     // Release a memory barrier
void      SDL_MemoryBarrierRelease  (void);                                     // Acquire a memory barrier
void      SDL_CPUPauseInstruction   (void);                                     // Execute hardware pause instruction, if supported
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
int       SDL_AtomicSet             (SDL_atomic_t *a, int v);                   // Set an atomic variable to a value.
int       SDL_AtomicGet             (SDL_atomic_t *a);                          // Get the value of an atomic variable.
int       SDL_AtomicAdd             (SDL_atomic_t *a, int v);                   // Add to an atomic variable.
SDL_bool  SDL_AtomicCASPtr          (void **a, void *oldval, void *newval);     // Set a pointer to a new value if it is currently an old value
void *    SDL_AtomicSetPtr          (void **a, void* v);                        // Set a pointer to a value atomically.
void *    SDL_AtomicGetPtr          (void **a);                                 // Get the value of a pointer atomically.


//
//
//
//
//
//
```



```

//|--------------------------------------------------------------------------
//| Timer Support (SDL_timer.h)
//|--------------------------------------------------------------------------
Uint64       SDL_GetTicks64              (void);                                                   // Get the number of milliseconds since
Uint32       SDL_GetTicks                (void);                                                   // Get the number of milliseconds since
Uint64       SDL_GetPerformanceCounter   (void);                                                   // Get the current value of the high res
Uint64       SDL_GetPerformanceFrequency (void);                                                   // Get the count per second of the high
void         SDL_Delay                   (Uint32 ms);                                              // Wait a specified number of millisecon
SDL_TimerID  SDL_AddTimer                (Uint32 interval, SDL_TimerCallback callback, void *param); // Call a callback function at a future
SDL_bool     SDL_RemoveTimer             (SDL_TimerID id);                                         // Remove a timer created with SDL_AddTi


//
//
//
//
//
//
```



```

//|--------------------------------------------------------------------------
//| Filesystem Paths (SDL_filesystem.h)
//|--------------------------------------------------------------------------
char *  SDL_GetBasePath  (void);                                  // Get the directory where the application was run from.
char *  SDL_GetPrefPath  (const char *org, const char *app);  // Get the user-and-app-specific path where files can be written.

//|--------------------------------------------------------------------------
//| File I/O Abstraction (SDL_rwops.h)
//|--------------------------------------------------------------------------
SDL_RWops *  SDL_RWFromFile      (const char *file, const char *mode);                              // Use this function to create a new SDL_RWop
SDL_RWops *  SDL_RWFromFP        (void *fp, SDL_bool autoclose);                                    // Use this function to create an SDL_RWops s
SDL_RWops *  SDL_RWFromMem       (void *mem, int size);                                             // Use this function to prepare a read-write
SDL_RWops *  SDL_RWFromConstMem  (const void *mem, int size);                                       // Use this function to prepare a read-only m
SDL_RWops *  SDL_AllocRW         (void);                                                            // Use this function to allocate an empty, un
void         SDL_FreeRW          (SDL_RWops *area);                                                 // Use this function to free an SDL_RWops str
Sint64       SDL_RWsize          (SDL_RWops *context);                                              // Use this function to get the size of the d
Sint64       SDL_RWseek          (SDL_RWops *context, Sint64 offset, int whence);                   // Seek within an SDL_RWops data stream.
Sint64       SDL_RWtell          (SDL_RWops *context);                                              // Determine the current read/write offset in
size_t       SDL_RWread          (SDL_RWops *context, void *ptr, size_t size, size_t maxnum);       // Read from a data source.
size_t       SDL_RWwrite         (SDL_RWops *context, const void *ptr, size_t size, size_t num);    // Write to an SDL_RWops data stream.
int          SDL_RWclose         (SDL_RWops *context);                                              // Close and free an allocated SDL_RWops stru
void *       SDL_LoadFile_RW     (SDL_RWops *src, size_t *datasize, int freesrc);                   // Load all the data from an SDL data stream.
void *       SDL_LoadFile        (const char *file, size_t *datasize);                              // Load all the data from a file path.
Uint8        SDL_ReadU8          (SDL_RWops *src);                                                  // Use this function to read a byte from an S
Uint16       SDL_ReadLE16        (SDL_RWops *src);                                                  // Use this function to read 16 bits of littl
Uint16       SDL_ReadBE16        (SDL_RWops *src);                                                  // Use this function to read 16 bits of big-e
Uint32       SDL_ReadLE32        (SDL_RWops *src);                                                  // Use this function to read 32 bits of littl
Uint32       SDL_ReadBE32        (SDL_RWops *src);                                                  // Use this function to read 32 bits of big-e
Uint64       SDL_ReadLE64        (SDL_RWops *src);                                                  // Use this function to read 64 bits of littl
Uint64       SDL_ReadBE64        (SDL_RWops *src);                                                  // Use this function to read 64 bits of big-e
size_t       SDL_WriteU8         (SDL_RWops *dst, Uint8 value);                                     // Use this function to write a byte to an SD
size_t       SDL_WriteLE16       (SDL_RWops *dst, Uint16 value);                                    // Use this function to write 16 bits in nati
size_t       SDL_WriteBE16       (SDL_RWops *dst, Uint16 value);                                    // Use this function to write 16 bits in nati
size_t       SDL_WriteLE32       (SDL_RWops *dst, Uint32 value);                                    // Use this function to write 32 bits in nati
size_t       SDL_WriteBE32       (SDL_RWops *dst, Uint32 value);                                    // Use this function to write 32 bits in nati
size_t       SDL_WriteBE64       (SDL_RWops *dst, Uint64 value);                                    // Use this function to write 64 bits in nati
size_t       SDL_WriteLE64       (SDL_RWops *dst, Uint64 value);                                    // Use this function to write 64 bits in nati


//
//
//
```

```c
//
//
//

//|-----------------------------------------------------------------------------
//| Shared Object Loading and Function Lookup (SDL_loadso.h)
//|-----------------------------------------------------------------------------
void *  SDL_LoadObject    (const char *sofile);             // Dynamically load a shared object.
void *  SDL_LoadFunction  (void *handle, const char *name);  // Look up the address of the named function in a shared object.
void    SDL_UnloadObject  (void *handle);                   // Unload a shared object from memory.


//
//
//    PLATFORM / CPU
//
//
//

//|-----------------------------------------------------------------------------
//| Platform Detection (SDL_platform.h)
//|-----------------------------------------------------------------------------
const char *  SDL_GetPlatform  (void);  // Get the name of the platform.

//|-----------------------------------------------------------------------------
//| CPU Feature Detection (SDL_cpuinfo.h)
//|-----------------------------------------------------------------------------
int       SDL_GetCPUCount          (void);                          // Get the number of CPU cores available.
int       SDL_GetCPUCacheLineSize  (void);                          // Determine the L1 cache line size of the CPU.
SDL_bool  SDL_HasRDTSC             (void);                          // Determine whether the CPU has the RDTSC instruction.
SDL_bool  SDL_HasAltiVec           (void);                          // Determine whether the CPU has AltiVec features.
SDL_bool  SDL_HasMMX               (void);                          // Determine whether the CPU has MMX features.
SDL_bool  SDL_Has3DNow             (void);                          // Determine whether the CPU has 3DNow! features.
SDL_bool  SDL_HasSSE               (void);                          // Determine whether the CPU has SSE features.
SDL_bool  SDL_HasSSE2              (void);                          // Determine whether the CPU has SSE2 features.
SDL_bool  SDL_HasSSE3              (void);                          // Determine whether the CPU has SSE3 features.
SDL_bool  SDL_HasSSE41             (void);                          // Determine whether the CPU has SSE4.1 features.
SDL_bool  SDL_HasSSE42             (void);                          // Determine whether the CPU has SSE4.2 features.
SDL_bool  SDL_HasAVX               (void);                          // Determine whether the CPU has AVX features.
SDL_bool  SDL_HasAVX2              (void);                          // Determine whether the CPU has AVX2 features.
SDL_bool  SDL_HasAVX512F           (void);                          // Determine whether the CPU has AVX-512F (foundation) features.
SDL_bool  SDL_HasARMSIMD           (void);                          // Determine whether the CPU has ARM SIMD (ARMv6) features.
SDL_bool  SDL_HasNEON              (void);                          // Determine whether the CPU has NEON (ARM SIMD) features.
SDL_bool  SDL_HasLSX               (void);                          // Determine whether the CPU has LSX (LOONGARCH SIMD) features.
SDL_bool  SDL_HasLASX              (void);                          // Determine whether the CPU has LASX (LOONGARCH SIMD) features.
int       SDL_GetSystemRAM         (void);                          // Get the amount of RAM configured in the system.
size_t    SDL_SIMDGetAlignment     (void);                          // Report the alignment this system needs for SIMD allocations.
void *    SDL_SIMDAlloc            (const size_t len);              // Allocate memory in a SIMD-friendly way.
void *    SDL_SIMDRealloc          (void *mem, const size_t len);   // Reallocate memory obtained from SDL_SIMDAlloc
void      SDL_SIMDFree             (void *ptr);                     // Deallocate memory obtained from SDL_SIMDAlloc

//|-----------------------------------------------------------------------------
//| Byte Order and Byte Swapping (SDL_endian.h)
//|-----------------------------------------------------------------------------
Uint16  SDL_Swap16       (Uint16 x);  // Unconditionally byte swap the provided data.
Uint32  SDL_Swap32       (Uint32 x);  // Unconditionally byte swap the provided data.
Uint64  SDL_Swap64       (Uint64 x);  // Unconditionally byte swap the provided data.
float   SDL_SwapFloat    (float x);   // Unconditionally byte swap the provided data.

Uint16  SDL_SwapLE16     (Uint16 X);  // Byteswap item from the specified endianness to the native endianness if necessary.
Uint32  SDL_SwapLE32     (Uint32 X);  // Byteswap item from the specified endianness to the native endianness if necessary.
Uint64  SDL_SwapLE64     (Uint64 X);  // Byteswap item from the specified endianness to the native endianness if necessary.
float   SDL_SwapFloatLE  (float X);   // Byteswap item from the specified endianness to the native endianness if necessary.
Uint16  SDL_SwapBE16     (Uint16 X);  // Byteswap item from the specified endianness to the native endianness if necessary.
Uint32  SDL_SwapBE32     (Uint32 X);  // Byteswap item from the specified endianness to the native endianness if necessary.
Uint64  SDL_SwapBE64     (Uint64 X);  // Byteswap item from the specified endianness to the native endianness if necessary.
float   SDL_SwapFloatBE  (float X);   // Byteswap item from the specified endianness to the native endianness if necessary.

//|-----------------------------------------------------------------------------
//| Bit Manipulation (SDL_bits.h)
//|-----------------------------------------------------------------------------
int       SDL_MostSignificantBitIndex32 (Uint32 x);  // Get the index of the most significant bit. Undefined when called with 0.
SDL_bool  SDL_HasExactlyOneBitSet32     (Uint32 x);  // Returns true if integer has exactly one bit set.


//
//
//    POWER MANAGEMENT
//
//
//

//|-----------------------------------------------------------------------------
//| Power Management Status (SDL_power.h)
//|-----------------------------------------------------------------------------
SDL_PowerState  SDL_GetPowerInfo  (int *seconds, int *percent);  // Get the current power supply details.


//
//
//    ADDITIONAL FUNCTION
//
//
//

//|-----------------------------------------------------------------------------
//| Platform-specific Functionality (SDL_system.h)
//|-----------------------------------------------------------------------------

// Windows
//---------------------
void               SDL_SetWindowsMessageHook     SDL_WindowsMessageHook callback, void *userdata);         // Set a callback for every Windows
int                SDL_Direct3D9GetAdapterIndex  (int displayIndex);                                      // Get the D3D9 adapter index that m
IDirect3DDevice9*  SDL_RenderGetD3D9Device       SDL_Renderer *renderer);                                 // Get the D3D9 device associated wi
ID3D11Device*      SDL_RenderGetD3D11Device      SDL_Renderer *renderer);                                 // Get the D3D11 device associated w
ID3D12Device*      SDL_RenderGetD3D12Device      SDL_Renderer *renderer);                                 // Get the D3D12 device associated w
SDL_bool           SDL_DXGIGetOutputInfo         (int displayIndex, int *adapterIndex, int *outputIndex);  // Get the DXGI Adapter and Output i
int                SDL_GDKGetTaskQueue           (XTaskQueueHandle *outTaskQueue);                        // Gets a reference to the global as

// Linux
//---------------------
int  SDL_LinuxSetThreadPriority           (Sint64 threadID, int priority);                      // Sets the UNIX nice value for a thread.
int  SDL_LinuxSetThreadPriorityAndPolicy  (Sint64 threadID, int sdlPriority, int schedPolicy);  // Sets the priority (not nice level) and sched

// iOS
//---------------------
int   SDL_iPhoneSetAnimationCallback                  (SDL_Window *window, int interval,
                                                       void (*callback)(void*), void *callbackParam);  // Use this function to set the animation
void  SDL_iPhoneSetEventPump                          (SDL_bool enabled);                              // Use this function to enable or disable

void  SDL_OnApplicationWillTerminate                  (void);   // Used by iOS application delegates to notify SDL about state changes.
void  SDL_OnApplicationDidReceiveMemoryWarning        (void);   // Used by iOS application delegates to notify SDL about state changes.
void  SDL_OnApplicationWillResignActive               (void);   // Used by iOS application delegates to notify SDL about state changes.
void  SDL_OnApplicationDidEnterBackground             (void);   // Used by iOS application delegates to notify SDL about state changes.
void  SDL_OnApplicationWillEnterForeground            (void);   // Used by iOS application delegates to notify SDL about state changes.
void  SDL_OnApplicationDidBecomeActive                (void);   // Used by iOS application delegates to notify SDL about state changes.
void  SDL_OnApplicationDidChangeStatusBarOrientation  (void);   // Used by iOS application delegates to notify SDL about state changes. (*only*

// Android
//----------------------
void *        SDL_AndroidGetJNIEnv                (void);  // Get the Android Java Native Interface Environment of the current thread.
void *        SDL_AndroidGetActivity              (void);  // Retrieve the Java instance of the Android activity class.
int           SDL_GetAndroidSDKVersion            (void);  // Query Android API level of the current device.
SDL_bool      SDL_IsAndroidTV                     (void);  // Query if the application is running on Android TV.
SDL_bool      SDL_IsChromebook                    (void);  // Query if the application is running on a Chromebook.
SDL_bool      SDL_IsDeXMode                       (void);  // Query if the application is running on a Samsung DeX docking station.
void          SDL_AndroidBackButton               (void);  // Trigger the Android system back button behavior.
const char *  SDL_AndroidGetInternalStoragePath   (void);  // Get the path used for internal storage for this application.
int           SDL_AndroidGetExternalStorageState  (void);  // Get the current state of external storage.
const char *  SDL_AndroidGetExternalStoragePath   (void);  // Get the path used for external storage for this application.

SDL_bool      SDL_AndroidRequestPermission        (const char *permission);                // Request permissions at runtime.
int           SDL_AndroidShowToast                (const char* message, int duration,
                                                   int gravity, int xoffset, int yoffset);  // Shows an Android toast notification.
int           SDL_AndroidSendMessage              (Uint32 command, int param);             // Send a user command to SDLActivity.

// WinRT (Windows Phone)
//----------------------
const wchar_t *         SDL_WinRTGetFSPathUNICODE  (SDL_WinRT_Path pathType);  // Retrieve a WinRT defined path on the local file system.
const char *            SDL_WinRTGetFSPathUTF8     (SDL_WinRT_Path pathType);  // Retrieve a WinRT defined path on the local file system.
SDL_WinRT_DeviceFamily  SDL_WinRTGetDeviceFamily   (void);                     // Detects the device family of WinRT platform at runtime.

// Misc.
//----------------------
SDL_bool  SDL_IsTablet  (void);  // Query if the current device is a tablet.

//|---------------------------------------------------------------------------
//| Standard Library Functionality (SDL_stdinc.h)
//|---------------------------------------------------------------------------

// Memory
//----------------------
void *  SDL_malloc                      (size_t size);
void *  SDL_calloc                      (size_t nmemb, size_t size);
void *  SDL_realloc                     (void *mem, size_t size);
void    SDL_free                        (void *mem);
void    SDL_GetOriginalMemoryFunctions  (SDL_malloc_func *malloc_func, SDL_calloc_func *calloc_func,
                                        SDL_realloc_func *realloc_func, SDL_free_func *free_func);    // Get the original set of SDL memory fun
void    SDL_GetMemoryFunctions          (SDL_malloc_func *malloc_func, SDL_calloc_func *calloc_func,
                                        SDL_realloc_func *realloc_func, SDL_free_func *free_func);    // Get the current set of SDL memory func
int     SDL_SetMemoryFunctions          (SDL_malloc_func malloc_func, SDL_calloc_func calloc_func,
                                        SDL_realloc_func realloc_func, SDL_free_func free_func);      // Replace SDL's memory allocation functi
int     SDL_GetNumAllocations           (void);                                                       // Get the number of outstanding (unfreed

// Environment Variables
//----------------------
char *  SDL_getenv  (const char *name);
int     SDL_setenv  (const char *name, const char *value, int overwrite);

// Sort/search
//----------------------
void    SDL_qsort    (void *base, size_t nmemb, size_t size, int (*compare) (const void *, const void *));
void *  SDL_bsearch  (const void *key, const void *base, size_t nmemb, size_t size, int (*compare) (const void *, const void *));

// Strings
//----------------------
int            SDL_isalpha      (int x);
int            SDL_isalnum      (int x);
int            SDL_isblank      (int x);
int            SDL_iscntrl      (int x);
int            SDL_isdigit      (int x);
int            SDL_isxdigit     (int x);
int            SDL_ispunct      (int x);
int            SDL_isspace      (int x);
int            SDL_isupper      (int x);
int            SDL_islower      (int x);
int            SDL_isprint      (int x);
int            SDL_isgraph      (int x);
int            SDL_toupper      (int x);
int            SDL_tolower      (int x);
Uint16         SDL_crc16        (Uint16 crc, const void *data, size_t len);
Uint32         SDL_crc32        (Uint32 crc, const void *data, size_t len);
void *         SDL_memset       (SDL_OUT_BYTECAP(len) void *dst, int c, size_t len);
void *         SDL_memcpy       (SDL_OUT_BYTECAP(len) void *dst, SDL_IN_BYTECAP(len) const void *src, size_t len);
void *         SDL_memcpy4      (SDL_OUT_BYTECAP(dwords*4) void *dst, SDL_IN_BYTECAP(dwords*4) const void *src, size_t dwords);
void *         SDL_memmove      (SDL_OUT_BYTECAP(len) void *dst, SDL_IN_BYTECAP(len) const void *src, size_t len);
int            SDL_memcmp       (const void *s1, const void *s2, size_t len);
size_t         SDL_wcslen       (const wchar_t *wstr);
size_t         SDL_wcslcpy      (SDL_OUT_Z_CAP(maxlen) wchar_t *dst, const wchar_t *src, size_t maxlen);
size_t         SDL_wcslcat      (SDL_INOUT_Z_CAP(maxlen) wchar_t *dst, const wchar_t *src, size_t maxlen);
wchar_t *      SDL_wcsdup       (const wchar_t *wstr);
wchar_t *      SDL_wcsstr       (const wchar_t *haystack, const wchar_t *needle);
int            SDL_wcscmp       (const wchar_t *str1, const wchar_t *str2);
int            SDL_wcsncmp      (const wchar_t *str1, const wchar_t *str2, size_t maxlen);
int            SDL_wcscasecmp   (const wchar_t *str1, const wchar_t *str2);
int            SDL_wcsncasecmp  (const wchar_t *str1, const wchar_t *str2, size_t len);
size_t         SDL_strlen       (const char *str);
size_t         SDL_strlcpy      (SDL_OUT_Z_CAP(maxlen) char *dst, const char *src, size_t maxlen);
size_t         SDL_utf8strlcpy  (SDL_OUT_Z_CAP(dst_bytes) char *dst, const char *src, size_t dst_bytes);
size_t         SDL_strlcat      (SDL_INOUT_Z_CAP(maxlen) char *dst, const char *src, size_t maxlen);
char *         SDL_strdup       (const char *str);
char *         SDL_strrev       (char *str);
char *         SDL_strupr       (char *str);
char *         SDL_strlwr       (char *str);
char *         SDL_strchr       (const char *str, int c);
char *         SDL_strrchr      (const char *str, int c);
char *         SDL_strstr       (const char *haystack, const char *needle);
char *         SDL_strcasestr   (const char *haystack, const char *needle);
char *         SDL_strtokr      (char *s1, const char *s2, char **saveptr);
size_t         SDL_utf8strlen   (const char *str);
size_t         SDL_utf8strnlen  (const char *str, size_t bytes);
char *         SDL_itoa         (int value, char *str, int radix);
char *         SDL_uitoa        (unsigned int value, char *str, int radix);
char *         SDL_ltoa         (long value, char *str, int radix);
char *         SDL_ultoa        (unsigned long value, char *str, int radix);
char *         SDL_lltoa        (Sint64 value, char *str, int radix);
char *         SDL_ulltoa       (Uint64 value, char *str, int radix);
int            SDL_atoi         (const char *str);
double         SDL_atof         (const char *str);
long           SDL_strtol       (const char *str, char **endp, int base);
unsigned long  SDL_strtoul      (const char *str, char **endp, int base);
Sint64         SDL_strtoll      (const char *str, char **endp, int base);
Uint64         SDL_strtoull     (const char *str, char **endp, int base);
double         SDL_strtod       (const char *str, char **endp);
int            SDL_strcmp       (const char *str1, const char *str2);
int            SDL_strncmp      (const char *str1, const char *str2, size_t maxlen);
int            SDL_strcasecmp   (const char *str1, const char *str2);
int            SDL_strncasecmp  (const char *str1, const char *str2, size_t len);
int            SDL_sscanf       (const char *text, SDL_SCANF_FORMAT_STRING const char *fmt, ...);
int            SDL_vsscanf      (const char *text, const char *fmt, va_list ap);
int            SDL_snprintf     (SDL_OUT_Z_CAP(maxlen) char *text, size_t maxlen, SDL_PRINTF_FORMAT_STRING const char *fmt, ...);
int            SDL_vsnprintf    (SDL_OUT_Z_CAP(maxlen) char *text, size_t maxlen, const char *fmt, va_list ap);
int            SDL_asprintf     (char **strp, SDL_PRINTF_FORMAT_STRING const char *fmt, ...);
int            SDL_vasprintf    (char **strp, const char *fmt, va_list ap);
```

```

// Math
//---------------------
int     SDL_abs        (int x);
double  SDL_acos       (double x);
float   SDL_acosf      (float x);
double  SDL_asin       (double x);
float   SDL_asinf      (float x);
double  SDL_atan       (double x);
float   SDL_atanf      (float x);
double  SDL_atan2      (double y, double x);
float   SDL_atan2f     (float y, float x);
double  SDL_ceil       (double x);
float   SDL_ceilf      (float x);
double  SDL_copysign   (double x, double y);
float   SDL_copysignf  (float x, float y);
double  SDL_cos        (double x);
float   SDL_cosf       (float x);
double  SDL_exp        (double x);
float   SDL_expf       (float x);
double  SDL_fabs       (double x);
float   SDL_fabsf      (float x);
double  SDL_floor      (double x);
float   SDL_floorf     (float x);
double  SDL_trunc      (double x);
float   SDL_truncf     (float x);
double  SDL_fmod       (double x, double y);
float   SDL_fmodf      (float x, float y);
double  SDL_log        (double x);
float   SDL_logf       (float x);
double  SDL_log10      (double x);
float   SDL_log10f     (float x);
double  SDL_pow        (double x, double y);
float   SDL_powf       (float x, float y);
double  SDL_round      (double x);
float   SDL_roundf     (float x);
long    SDL_lround     (double x);
long    SDL_lroundf    (float x);
double  SDL_scalbn     (double x, int n);
float   SDL_scalbnf    (float x, int n);
double  SDL_sin        (double x);
float   SDL_sinf       (float x);
double  SDL_sqrt       (double x);
float   SDL_sqrtf      (float x);
double  SDL_tan        (double x);
float   SDL_tanf       (float x);

// Unicode
//---------------------
SDL_iconv_t  SDL_iconv_open         (const char *tocode, const char *fromcode);
int          SDL_iconv_close        (SDL_iconv_t cd);
size_t       SDL_iconv              (SDL_iconv_t cd, const char **inbuf, size_t *inbytesleft, char **outbuf, size_t *outbytesleft);
char *       SDL_iconv_string       (const char *tocode, const char *fromcode, const char *inbuf, size_t inbytesleft); // This function convert
char *       SDL_iconv_utf8_locale  (const char *inbuf);
char *       SDL_iconv_utf8_ucs2    (const char *inbuf);
char *       SDL_iconv_utf8_ucs4    (const char *inbuf);
char *       SDL_iconv_wchar_utf8   (const char *inbuf);
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
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

Sign in with GitHub