Simple Directmedia Layer 3.1.2 Reference Manual

The SDL Developers

May 5, 2024

Chapter 1

SDL Functions

SDL_acos

Compute the arc cosine of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_acos(double x);
```

Function Parameters

x	floating point value

Return Value

Returns arc cosine of `x`, in radians

Remarks

The definition of `y = acos(x)` is `x = cos(y)`.

Domain: `-1 <= x <= 1`

Range: `0 <= y <= Pi`

This function operates on double-precision floating point values, use SDL_acosf for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Code Examples

```
/* acos(x) = Pi/2 - asin(x) */
SDL_Log("acos(0):\t\t%f", SDL_acos(0.));
SDL_Log("Pi/2 - asin(0):\t%f", SDL_PI_D / 2 - SDL_asin(0));

/* acos(-x) = Pi - acos(x) */
SDL_Log("acos(-(-1)):\t%f", SDL_acos(-(-1)));
SDL_Log("Pi - acos(-1):\t%f", SDL_PI_D - SDL_acos(-1));
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_acosf
- SDL_asin
- SDL_cos

SDL_acosf

Compute the arc cosine of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_acosf(float x);
```

Function Parameters

x	floating point value.

Return Value

Returns arc cosine of `x`, in radians

Remarks

The definition of `y = acos(x)` is `x = cos(y)`.

Domain: `-1 <= x <= 1`

Range: `0 <= y <= Pi`

This function operates on single-precision floating point values, use SDL_acos for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_acos
- SDL_asinf
- SDL_cosf

SDL_AcquireCameraFrame

Acquire a frame.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
SDL_Surface * SDL_AcquireCameraFrame(SDL_Camera *camera, Uint64
    *timestampNS);
```

Function Parameters

camera	opened camera device
timestampNS	a pointer filled in with the frame's timestamp, or 0 on error. Can be NULL.

Return Value

Returns A new frame of video on success, NULL if none is currently available.

Remarks

The frame is a memory pointer to the image data, whose size and format are given by the spec requested when opening the device.

This is a non blocking API. If there is a frame available, a non-NULL surface is returned, and timestampNS will be filled with a non-zero value.

Note that an error case can also return NULL, but a NULL by itself is normal and just signifies that a new frame is not yet available. Note that even if a camera device fails outright (a USB camera is unplugged while in use, etc), SDL will send an event separately to notify the app, but continue to provide blank frames at ongoing intervals until SDL_CloseCamera() is called, so real failure here is almost always an out of memory condition.

After use, the frame should be released with SDL_ReleaseCameraFrame(). If you don't do this, the system may stop providing more video!

Do not call SDL_FreeSurface() on the returned surface! It must be given back to the camera subsystem with SDL_ReleaseCameraFrame!

If the system is waiting for the user to approve access to the camera, as some platforms require, this will return NULL (no frames available); you should either wait for an SDL_EVENT_CAMERA_DEVICE_APPROVED (or SDL_EVENT_CAMERA_DEVICE_DENIED) event, or poll SDL_IsCameraApproved() occasionally until it returns non-zero.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ReleaseCameraFrame

SDL_AddEventWatch

Add a callback to be triggered when an event is added to the event queue.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
int SDL_AddEventWatch(SDL_EventFilter filter, void *userdata);
```

Function Parameters

filter	an SDL_EventFilter function to call when an event happens.
userdata	a pointer that is passed to `filter`

Return Value

Returns 0 on success, or a negative error code on failure; call SDL_GetError() for more information.

Remarks

`filter` will be called when an event happens, and its return value is ignored. **WARNING**: Be very careful of what you do in the event filter function, as it may run in a different thread!

If the quit event is generated by a signal (e.g. SIGINT), it will bypass the internal queue and be delivered to the watch callback immediately, and arrive at the next event poll.

Note: the callback is called for events posted by the user through SDL_PushEvent(), but not for disabled events, nor for events by a filter callback set with SDL_SetEventFilter(), nor for events posted by the user through SDL_PeepEvents().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DelEventWatch
- SDL_SetEventFilter

SDL_AddGamepadMapping

Add support for gamepads that SDL is unaware of or change the binding of an existing gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_AddGamepadMapping(const char *mapping);
```

Function Parameters

mapping	the mapping string

Return Value

Returns 1 if a new mapping is added, 0 if an existing mapping is updated, -1 on error; call SDL_GetError() for more information.

Remarks

The mapping string has the format "GUID,name,mapping", where GUID is the string value from SDL_GetJoystickGUIDString(), name is the human readable string for the device and mappings are gamepad mappings to joystick ones. Under Windows there is a reserved GUID of "xinput" that covers all XInput devices. The mapping format for joystick is:

- `bX`: a joystick button, index X
- `hX.Y`: hat X with value Y
- `aX`: axis X of the joystick

Buttons can be used as a gamepad axes and vice versa.
This string shows an example of a valid mapping for a gamepad:

```
"341a3608000000000000504944564944,Afterglow PS3
    Controller,a:b1,b:b2,y:b3,x:b0,start:b9,guide:b12,back:b8,dpup:h0.1,dpleft:h0.8,dpdown:h0.4,
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadMapping
- SDL_GetGamepadMappingForGUID

SDL_AddGamepadMappingsFromFile

Load a set of gamepad mappings from a file.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_AddGamepadMappingsFromFile(const char *file);
```

Function Parameters

file	the mappings file to load

Return Value

Returns the number of mappings added or -1 on error; call SDL_GetError() for more information.

Remarks

You can call this function several times, if needed, to load different database files.

If a new mapping is loaded for an already known gamepad GUID, the later version will overwrite the one currently loaded.

Mappings not belonging to the current platform or with no platform field specified will be ignored (i.e. mappings for Linux will be ignored in Windows, etc).

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AddGamepadMapping
- SDL_AddGamepadMappingsFromIO
- SDL_GetGamepadMapping
- SDL_GetGamepadMappingForGUID

SDL_AddGamepadMappingsFromIO

Load a set of gamepad mappings from an SDL_IOStream.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_AddGamepadMappingsFromIO(SDL_IOStream *src, SDL_bool closeio);
```

Function Parameters

src	the data stream for the mappings to be added
closeio	if SDL_TRUE, calls SDL_CloseIO() on `src` before returning, even in the case of an error

Return Value

Returns the number of mappings added or -1 on error; call SDL_GetError() for more information.

Remarks

You can call this function several times, if needed, to load different database files.

If a new mapping is loaded for an already known gamepad GUID, the later version will overwrite the one currently loaded.

Mappings not belonging to the current platform or with no platform field specified will be ignored (i.e. mappings for Linux will be ignored in Windows, etc).

This function will load the text database entirely in memory before processing it, so take this into consideration if you are in a memory constrained environment.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AddGamepadMapping
- SDL_AddGamepadMappingsFromFile

- SDL_GetGamepadMapping
- SDL_GetGamepadMappingForGUID

SDL_AddHintCallback

Add a function to watch a particular hint.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
int SDL_AddHintCallback(const char *name,
                        SDL_HintCallback callback,
                        void *userdata);
```

Function Parameters

name	the hint to watch
callback	An SDL_HintCallback function that will be called when the hint value changes
userdata	a pointer to pass to the callback function

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DelHintCallback

SDL_AddTimer

Call a callback function at a future time.

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
SDL_TimerID SDL_AddTimer(Uint32 interval,
                         SDL_TimerCallback callback,
                         void *param);
```

Function Parameters

interval	the timer delay, in milliseconds, passed to `callback`
callback	the SDL_TimerCallback function to call when the specified `interval` elapses
param	a pointer that is passed to `callback`

Return Value

Returns a timer ID or 0 if an error occurs; call SDL_GetError() for more information.

Remarks

If you use this function, you must pass `SDL_INIT_TIMER` to SDL_Init().

The callback function is passed the current timer interval and the user supplied parameter from the SDL_AddTimer() call and should return the next timer interval. If the value returned from the callback is 0, the timer is canceled.

The callback is run on a separate thread.

Timers take into account the amount of time it took to execute the callback. For example, if the callback took 250 ms to execute and returned 1000 (ms), the timer would only wait another 750 ms before its next iteration.

Timing may be inexact due to OS scheduling. Be sure to note the current time with SDL_GetTicksNS() or SDL_GetPerformanceCounter() in case your callback needs to adjust for variances.

Code Examples

```
Uint32 my_callbackfunc_1(Uint32 interval, void *param) {
    SDL_Event event;
    SDL_UserEvent userevent;
```

```
    /* In this example, our callback pushes an SDL_EVENT_USER event
    into the queue, and causes our callback to be called again at the
    same interval: */

    userevent.type = SDL_EVENT_USER;
    userevent.code = 0;
    userevent.data1 = NULL;
    userevent.data2 = NULL;

    event.type = SDL_EVENT_USER;
    event.user = userevent;

    SDL_PushEvent(&event);
    return interval;
}

int my_callback_param;

/* Start the timer; the callback below will be executed after the delay
    */
Uint32 delay = ((33 + 5) / 10) * 10; /* To round it down to the nearest
    10 ms */
SDL_TimerID my_timer_id = SDL_AddTimer(delay, my_callbackfunc_1,
    &my_callback_param);

/* ... */
```

Note that it is possible to avoid the multithreading problems with SDL timers by giving to `userevent.data1` the address of a function you want to be executed and to `userevent.data2` its params, and then deal with it in the event loop.

```
void my_function(void *);

/* with the same code as before: */
Uint32 my_callbackfunc_2(Uint32 interval, void *param) {
    SDL_Event event;
    SDL_UserEvent userevent;

    /* In this example, our callback pushes a function
    into the queue, and causes our callback to be called again at the
    same interval: */

    userevent.type = SDL_EVENT_USER;
    userevent.code = 0;
    userevent.data1 = &my_function;
    userevent.data2 = param;

    event.type = SDL_EVENT_USER;
```

```
    event.user = userevent;

    SDL_PushEvent(&event);
    return(interval);
}

/* Now the event loop */
SDL_Event event;
while (SDL_PollEvent (&event))
{
    switch(event.type)
    {
        case SDL_EVENT_USER: {
            /* and now we can call the function we wanted to call in the
                timer but couldn't because of the multithreading
                problems */
            void (*p) (void*) = event.user.data1;
            p(event.user.data2);
            break;
        }
        /* ... */
    }
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RemoveTimer

SDL_AddVulkanRenderSemaphores

Add a set of synchronization semaphores for the current frame.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_AddVulkanRenderSemaphores(SDL_Renderer *renderer, Uint32
    wait_stage_mask, Sint64 wait_semaphore, Sint64 signal_semaphore);
```

Function Parameters

renderer	the rendering context
wait_stage_mask	the VkPipelineStageFlags for the wait
wait_semaphore	a VkSempahore to wait on before rendering the current frame, or 0 if not needed
signal_semaphore	a VkSempahore that SDL will signal when rendering for the current frame is complete, or 0 if not needed

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The Vulkan renderer will wait for `wait_semaphore` before submitting rendering commands and signal `signal_semaphore` after rendering commands are complete for this frame.

This should be called each frame that you want semaphore synchronization. The Vulkan renderer may have multiple frames in flight on the GPU, so you should have multiple semaphores that are used for synchronization. Querying SDL_PROP_RENDERER_VULKAN_SWAPCHAIN_IMAGE_COUNT_NUMBER will give you the maximum number of semaphores you'll need.

Version

This function is available since SDL 3.0.0.

SDL_aligned_alloc

Allocate memory aligned to a specific value.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
SDL_MALLOC void* SDL_aligned_alloc(size_t alignment, size_t size);
```

Function Parameters

alignment	the alignment requested
size	the size to allocate

Return Value

Returns a pointer to the aligned memory

Remarks

If `alignment` is less than the size of `void *`, then it will be increased to match that.

The returned memory address will be a multiple of the alignment value, and the amount of memory allocated will be a multiple of the alignment value.

The memory returned by this function must be freed with SDL_aligned_free()

Version

This function is available since SDL 3.0.0.

See Also

- SDL_aligned_free

SDL_aligned_free

Free memory allocated by SDL_aligned_alloc().

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
void SDL_aligned_free(void *mem);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_aligned_alloc

SDL_AllocateEventMemory

Allocate dynamic memory for an SDL event.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
void * SDL_AllocateEventMemory(size_t size);
```

Function Parameters

size	the amount of memory to allocate

Return Value

Returns a pointer to the memory allocated or NULL on failure; call SDL_GetError() for more information.

Remarks

You can use this to allocate memory for user events that will be automatically freed after the event is processed.

Version

This function is available since SDL 3.0.0.

SDL_AndroidBackButton

Trigger the Android system back button behavior.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_AndroidBackButton(void);
```

Version

This function is available since SDL 3.0.0.

SDL_AndroidGetActivity

Retrieve the Java instance of the Android activity class.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void * SDL_AndroidGetActivity(void);
```

Return Value

Returns the jobject representing the instance of the Activity class of the Android application, or NULL on error.

Remarks

The prototype of the function in SDL's code actually declares a void* return type, even if the implementation returns a jobject. The rationale being that the SDL headers can avoid including jni.h.

The jobject returned by the function is a local reference and must be released by the caller. See the PushLocalFrame() and PopLocalFrame() or DeleteLocalRef() functions of the Java native interface:

https://docs.oracle.com/javase/1.5.0/docs/guide/jni/spec/functions.html

Code Examples

```
#include <SDL3/SDL.h>
#include <jni.h>

// This example requires C++ and a custom Java method named "void
    showHome()"

// Calls the void showHome() method of the Java instance of the activity.
void showHome(void)
{
    // retrieve the JNI environment.
    JNIEnv* env = (JNIEnv*)SDL_AndroidGetJNIEnv();

    // retrieve the Java instance of the SDLActivity
    jobject activity = (jobject)SDL_AndroidGetActivity();
```

```
    // find the Java class of the activity. It should be SDLActivity or
        a subclass of it.
    jclass clazz(env->GetObjectClass(activity));

    // find the identifier of the method to call
    jmethodID method_id = env->GetMethodID(clazz, "showHome", "()V");

    // effectively call the Java method
    env->CallVoidMethod(activity, method_id);

    // clean up the local references.
    env->DeleteLocalRef(activity);
    env->DeleteLocalRef(clazz);

    // Warning (and discussion of implementation details of SDL for
        Android):
    // Local references are automatically deleted if a native function
        called
    // from Java side returns. For SDL this native function is main()
        itself.
    // Therefore references need to be manually deleted because
        otherwise the
    // references will first be cleaned if main() returns (application
        exit).
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AndroidGetJNIEnv

SDL_AndroidGetExternalStoragePath

Get the path used for external storage for this application.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
const char * SDL_AndroidGetExternalStoragePath(void);
```

Return Value

Returns the path used for external storage for this application on success or NULL on failure; call SDL_GetError() for more information.

Remarks

This path is unique to your application, but is public and can be written to by other applications.

Your external storage path is typically: `/storage/sdcard0/Android/data/your.app.package/file`

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AndroidGetExternalStorageState

SDL_AndroidGetExternalStorageState

Get the current state of external storage.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_AndroidGetExternalStorageState(Uint32 *state);
```

Function Parameters

state	filled with the current state of external storage. 0 if external storage is currently unavailable.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The current state of external storage, a bitmask of these values: `SDL_ANDROID_EXTERNAL_STORAGE_READ`, `SDL_ANDROID_EXTERNAL_STORAGE_WRITE`.

If external storage is currently unavailable, this will return 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AndroidGetExternalStoragePath

SDL_AndroidGetInternalStoragePath

Get the path used for internal storage for this application.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
const char * SDL_AndroidGetInternalStoragePath(void);
```

Return Value

Returns the path used for internal storage or NULL on failure; call SDL_GetError() for more information.

Remarks

This path is unique to your application and cannot be written to by other applications.

Your internal storage path is typically: `/data/data/your.app.package/files`.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AndroidGetExternalStorageState

SDL_AndroidGetJNIEnv

Get the Android Java Native Interface Environment of the current thread.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void * SDL_AndroidGetJNIEnv(void);
```

Return Value

Returns a pointer to Java native interface object (JNIEnv) to which the current thread is attached, or 0 on error.

Remarks

This is the JNIEnv one needs to access the Java virtual machine from native code, and is needed for many Android APIs to be usable from C.

The prototype of the function in SDL's code actually declare a void* return type, even if the implementation returns a pointer to a JNIEnv. The rationale being that the SDL headers can avoid including jni.h.

Code Examples

```
#include <SDL3/SDL.h>
#include <jni.h>

// This example requires C++ and a custom Java method named "void
    showHome()"

// Calls the void showHome() method of the Java instance of the activity.
void showHome(void)
{
    // retrieve the JNI environment.
    JNIEnv* env = (JNIEnv*)SDL_AndroidGetJNIEnv();

    // retrieve the Java instance of the SDLActivity
    jobject activity = (jobject)SDL_AndroidGetActivity();

    // find the Java class of the activity. It should be SDLActivity or
        a subclass of it.
    jclass clazz(env->GetObjectClass(activity));
```

```
    // find the identifier of the method to call
    jmethodID method_id = env->GetMethodID(clazz, "showHome", "()V");

    // effectively call the Java method
    env->CallVoidMethod(activity, method_id);

    // clean up the local references.
    env->DeleteLocalRef(activity);
    env->DeleteLocalRef(clazz);

    // Warning (and discussion of implementation details of SDL for
        Android):
    // Local references are automatically deleted if a native function
        called
    // from Java side returns. For SDL this native function is main()
        itself.
    // Therefore references need to be manually deleted because
        otherwise the
    // references will first be cleaned if main() returns (application
        exit).
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AndroidGetActivity

SDL_AndroidRequestPermission

Request permissions at runtime, asynchronously.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_AndroidRequestPermission(const char *permission,
    SDL_AndroidRequestPermissionCallback cb, void *userdata);
```

Function Parameters

permission	The permission to request.
cb	The callback to trigger when the request has a response.
userdata	An app-controlled pointer that is passed to the callback.

Return Value

Returns zero if the request was submitted, -1 if there was an error submitting. The result of the request is only ever reported through the callback, not this return value.

Remarks

You do not need to call this for built-in functionality of SDL; recording from a microphone or reading images from a camera, using standard SDL APIs, will manage permission requests for you.

This function never blocks. Instead, the app-supplied callback will be called when a decision has been made. This callback may happen on a different thread, and possibly much later, as it might wait on a user to respond to a system dialog. If permission has already been granted for a specific entitlement, the callback will still fire, probably on the current thread and before this function returns.

If the request submission fails, this function returns -1 and the callback will NOT be called, but this should only happen in catastrophic conditions, like memory running out. Normally there will be a yes or no to the request through the callback.

Version

This function is available since SDL 3.0.0.

SDL_AndroidSendMessage

Send a user command to SDLActivity.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_AndroidSendMessage(Uint32 command, int param);
```

Function Parameters

command	user command that must be greater or equal to 0x8000
param	user parameter

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Override "boolean onUnhandledMessage(Message msg)" to handle the message.

Version

This function is available since SDL 3.0.0.

SDL_AndroidShowToast

Shows an Android toast notification.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_AndroidShowToast(const char* message, int duration, int gravity,
    int xoffset, int yoffset);
```

Function Parameters

message	text message to be shown
duration	0=short, 1=long
gravity	where the notification should appear on the screen.
xoffset	set this parameter only when gravity ¿=0
yoffset	set this parameter only when gravity ¿=0

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Toasts are a sort of lightweight notification that are unique to Android.

https://developer.android.com/guide/topics/ui/notifiers/toasts

Shows toast in UI thread.

For the `gravity` parameter, choose a value from here, or -1 if you don't have a preference:

https://developer.android.com/reference/android/view/Gravity

Version

This function is available since SDL 3.0.0.

SDL_AppEvent

App-implemented event entry point for SDL_MAIN_USE_CALLBACKS apps.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
int SDL_AppEvent(void *appstate, const SDL_Event *event);
```

Function Parameters

appstate	an optional pointer, provided by the app in SDL_AppInit.
event	the new event for the app to examine.

Return Value

Returns -1 to terminate with an error, 1 to terminate with success, 0 to continue.

Remarks

Apps implement this function when using SDL_MAIN_USE_CALLBACKS. If using a standard "main" function, you should not supply this.

This function is called as needed by SDL after SDL_AppInit returns 0; It is called once for each new event.

There is (currently) no guarantee about what thread this will be called from; whatever thread pushes an event onto SDL's queue will trigger this function. SDL is responsible for pumping the event queue between each call to SDL_AppIterate, so in normal operation one should only get events in a serial fashion, but be careful if you have a thread that explicitly calls SDL_PushEvent.

Events sent to this function are not owned by the app; if you need to save the data, you should copy it.

You do not need to free event data (such as the `file` string in SDL_EVENT_DROP_FILE), as SDL will free it once this function returns. Note that this is different than one might expect when using a standard "main" function!

This function should not go into an infinite mainloop; it should handle the provided event appropriately and return.

The `appstate` parameter is an optional pointer provided by the app during SDL_AppInit(). If the app never provided a pointer, this will be NULL.

If this function returns 0, the app will continue normal operation, receiving repeated calls to SDL_AppIterate and SDL_AppEvent for the life of the program. If this function returns ¡ 0, SDL will call SDL_AppQuit and terminate the process with an exit code that reports an error to the platform. If it returns ¿ 0, the

SDL calls SDL_AppQuit and terminates with an exit code that reports success to the platform.

Thread Safety

This function is not thread safe.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AppInit
- SDL_AppIterate

SDL_AppInit

App-implemented initial entry point for SDL_MAIN_USE_CALLBACKS apps.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
int SDL_AppInit(void **appstate, int argc, char *argv[]);
```

Function Parameters

appstate	a place where the app can optionally store a pointer for future use.
argc	The standard ANSI C main's argc; number of elements in `argv`
argv	The standard ANSI C main's argv; array of command line arguments.

Return Value

Returns -1 to terminate with an error, 1 to terminate with success, 0 to continue.

Remarks

Apps implement this function when using SDL_MAIN_USE_CALLBACKS. If using a standard "main" function, you should not supply this.

This function is called by SDL once, at startup. The function should initialize whatever is necessary, possibly create windows and open audio devices, etc. The `argc` and `argv` parameters work like they would with a standard "main" function.

This function should not go into an infinite mainloop; it should do any one-time setup it requires and then return.

The app may optionally assign a pointer to `*appstate`. This pointer will be provided on every future call to the other entry points, to allow application state to be preserved between functions without the app needing to use a global variable. If this isn't set, the pointer will be NULL in future entry points.

If this function returns 0, the app will proceed to normal operation, and will begin receiving repeated calls to SDL_AppIterate and SDL_AppEvent for the life of the program. If this function returns ¡ 0, SDL will call SDL_AppQuit and terminate the process with an exit code that reports an error to the platform. If it returns ¿ 0, the SDL calls SDL_AppQuit and terminates with an exit code that reports success to the platform.

Thread Safety

This function is not thread safe.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AppIterate
- SDL_AppEvent
- SDL_AppQuit

SDL_AppIterate

App-implemented iteration entry point for SDL_MAIN_USE_CALLBACKS apps.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
int SDL_AppIterate(void *appstate);
```

Function Parameters

appstate	an optional pointer, provided by the app in SDL_AppInit.

Return Value

Returns -1 to terminate with an error, 1 to terminate with success, 0 to continue.

Remarks

Apps implement this function when using SDL_MAIN_USE_CALLBACKS. If using a standard "main" function, you should not supply this.

This function is called repeatedly by SDL after SDL_AppInit returns 0. The function should operate as a single iteration the program's primary loop; it should update whatever state it needs and draw a new frame of video, usually.

On some platforms, this function will be called at the refresh rate of the display (which might change during the life of your app!). There are no promises made about what frequency this function might run at. You should use SDL's timer functions if you need to see how much time has passed since the last iteration.

There is no need to process the SDL event queue during this function; SDL will send events as they arrive in SDL_AppEvent, and in most cases the event queue will be empty when this function runs anyhow.

This function should not go into an infinite mainloop; it should do one iteration of whatever the program does and return.

The `appstate` parameter is an optional pointer provided by the app during SDL_AppInit(). If the app never provided a pointer, this will be NULL.

If this function returns 0, the app will continue normal operation, receiving repeated calls to SDL_AppIterate and SDL_AppEvent for the life of the program. If this function returns ¡ 0, SDL will call SDL_AppQuit and terminate the process with an exit code that reports an error to the platform. If it returns ¿ 0, the SDL calls SDL_AppQuit and terminates with an exit code that reports success to the platform.

Thread Safety

This function is not thread safe.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AppInit
- SDL_AppEvent

SDL_AppQuit

App-implemented deinit entry point for SDL_MAIN_USE_CALLBACKS apps.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
void SDL_AppQuit(void *appstate);
```

Function Parameters

appstate	an optional pointer, provided by the app in SDL_AppInit.

Remarks

Apps implement this function when using SDL_MAIN_USE_CALLBACKS. If using a standard "main" function, you should not supply this.

This function is called once by SDL before terminating the program.

This function will be called no matter what, even if SDL_AppInit requests termination.

This function should not go into an infinite mainloop; it should deinitialize any resources necessary, perform whatever shutdown activities, and return.

You do not need to call SDL_Quit() in this function, as SDL will call it after this function returns and before the process terminates, but it is safe to do so.

The `appstate` parameter is an optional pointer provided by the app during SDL_AppInit(). If the app never provided a pointer, this will be NULL. This function call is the last time this pointer will be provided, so any resources to it should be cleaned up here.

Thread Safety

This function is not thread safe.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AppInit

SDL_asin

Compute the arc sine of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_asin(double x);
```

Function Parameters

x	floating point value.

Return Value

Returns arc sine of `x`, in radians.

Remarks

The definition of `y = asin(x)` is `x = sin(y)`.

Domain: `-1 <= x <= 1`

Range: `-Pi/2 <= y <= Pi/2`

This function operates on double-precision floating point values, use SDL_asinf for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_asinf
- SDL_acos
- SDL_sin

SDL_asinf

Compute the arc sine of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_asinf(float x);
```

Function Parameters

x	floating point value.

Return Value

Returns arc sine of `x`, in radians.

Remarks

The definition of `y = asin(x)` is `x = sin(y)`.

Domain: `-1 <= x <= 1`

Range: `-Pi/2 <= y <= Pi/2`

This function operates on single-precision floating point values, use SDL_asin for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_asin
- SDL_acosf
- SDL_sinf

SDL_atan

Compute the arc tangent of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_atan(double x);
```

Function Parameters

x	floating point value.

Return Value

Returns arc tangent of of `x` in radians, or 0 if `x = 0`.

Remarks

The definition of `y = atan(x)` is `x = tan(y)`.

Domain: `-INF <= x <= INF`

Range: `-Pi/2 <= y <= Pi/2`

This function operates on double-precision floating point values, use SDL_atanf for single-precision floats.

To calculate the arc tangent of y / x, use SDL_atan2.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_atanf
- SDL_atan2
- SDL_tan

SDL_atan2

Compute the arc tangent of `y / x`, using the signs of x and y to adjust the result's quadrant.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_atan2(double y, double x);
```

Function Parameters

x	floating point value of the denominator (x coordinate).
y	floating point value of the numerator (y coordinate)

Return Value

Returns arc tangent of of `y / x` in radians, or, if `x = 0`, either `-Pi/2`, `0`, or `Pi/2`, depending on the value of `y`.

Remarks

The definition of `z = atan2(x, y)` is `y = x tan(z)`, where the quadrant of z is determined based on the signs of x and y.

Domain: `-INF <= x <= INF`, `-INF <= y <= INF`

Range: `-Pi/2 <= y <= Pi/2`

This function operates on double-precision floating point values, use SDL_atan2f for single-precision floats.

To calculate the arc tangent of a single value, use SDL_atan.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_atan2f
- SDL_atan

- SDL_tan

SDL_atan2f

Compute the arc tangent of `y / x`, using the signs of x and y to adjust the result's quadrant.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_atan2f(float y, float x);
```

Function Parameters

x	floating point value of the denominator (x coordinate).
y	floating point value of the numerator (y coordinate)

Return Value

Returns arc tangent of of `y / x` in radians, or, if `x = 0`, either `-Pi/2`, `0`, or `Pi/2`, depending on the value of `y`.

Remarks

The definition of `z = atan2(x, y)` is `y = x tan(z)`, where the quadrant of z is determined based on the signs of x and y.

Domain: `-INF <= x <= INF`, `-INF <= y <= INF`

Range: `-Pi/2 <= y <= Pi/2`

This function operates on single-precision floating point values, use SDL_atan2 for double-precision floats.

To calculate the arc tangent of a single value, use SDL_atanf.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_atan2f
- SDL_atan

- SDL_tan

SDL_atanf

Compute the arc tangent of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_atanf(float x);
```

Function Parameters

x	floating point value.

Return Value

Returns arc tangent of of `x` in radians, or 0 if `x = 0`

Remarks

The definition of `y = atan(x)` is `x = tan(y)`.

Domain: `-INF <= x <= INF`

Range: `-Pi/2 <= y <= Pi/2`

This function operates on single-precision floating point values, use SDL_atan for dboule-precision floats.

To calculate the arc tangent of y / x, use SDL_atan2f.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_atan
- SDL_atan2f
- SDL_tanf

SDL_AtomicAdd

Add to an atomic variable.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
int SDL_AtomicAdd(SDL_AtomicInt *a, int v);
```

Function Parameters

a	a pointer to an SDL_AtomicInt variable to be modified
v	the desired value to add

Return Value

Returns the previous value of the atomic variable.

Remarks

This function also acts as a full memory barrier. ***Note: If you don't know what this function is for, you shouldn't use it!***

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AtomicDecRef
- SDL_AtomicIncRef

SDL_AtomicCompareAndSwap

Set an atomic variable to a new value if it is currently an old value.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
SDL_bool SDL_AtomicCompareAndSwap(SDL_AtomicInt *a, int oldval, int
    newval);
```

Function Parameters

a	a pointer to an SDL_AtomicInt variable to be modified
oldval	the old value
newval	the new value

Return Value

Returns SDL_TRUE if the atomic variable was set, SDL_FALSE otherwise.

Remarks

Note: If you don't know what this function is for, you shouldn't use it!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AtomicCompareAndSwapPointer

SDL_AtomicCompareAndSwapPointer

Set a pointer to a new value if it is currently an old value.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
SDL_bool SDL_AtomicCompareAndSwapPointer(void **a, void *oldval, void
    *newval);
```

Function Parameters

a	a pointer to a pointer
oldval	the old pointer value
newval	the new pointer value

Return Value

Returns SDL_TRUE if the pointer was set, SDL_FALSE otherwise.

Remarks

Note: If you don't know what this function is for, you shouldn't use it!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AtomicCompareAndSwap
- SDL_AtomicGetPtr
- SDL_AtomicSetPtr

SDL_AtomicGet

Get the value of an atomic variable.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
int SDL_AtomicGet(SDL_AtomicInt *a);
```

Function Parameters

a	a pointer to an SDL_AtomicInt variable

Return Value

Returns the current value of an atomic variable.

Remarks

Note: If you don't know what this function is for, you shouldn't use it!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AtomicSet

SDL_AtomicGetPtr

Get the value of a pointer atomically.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
void* SDL_AtomicGetPtr(void **a);
```

Function Parameters

a	a pointer to a pointer

Return Value

Returns the current value of a pointer.

Remarks

Note: If you don't know what this function is for, you shouldn't use it!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AtomicCompareAndSwapPointer
- SDL_AtomicSetPtr

SDL_AtomicSet

Set an atomic variable to a value.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
int SDL_AtomicSet(SDL_AtomicInt *a, int v);
```

Function Parameters

a	a pointer to an SDL_AtomicInt variable to be modified
v	the desired value

Return Value

Returns the previous value of the atomic variable.

Remarks

This function also acts as a full memory barrier. ***Note: If you don't know what this function is for, you shouldn't use it!***

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AtomicGet

SDL_AtomicSetPtr

Set a pointer to a value atomically.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
void* SDL_AtomicSetPtr(void **a, void* v);
```

Function Parameters

a	a pointer to a pointer
v	the desired pointer value

Return Value

Returns the previous value of the pointer.

Remarks

Note: If you don't know what this function is for, you shouldn't use it!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AtomicCompareAndSwapPointer
- SDL_AtomicGetPtr

SDL_AttachVirtualJoystick

Attach a new virtual joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickID SDL_AttachVirtualJoystick(SDL_JoystickType type,
                                         int naxes,
                                         int nbuttons,
                                         int nhats);
```

Function Parameters

type	type of joystick
naxes	number of axes
nbuttons	number of buttons
nhats	number of hats

Return Value

Returns the joystick instance ID, or 0 if an error occurred; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AttachVirtualJoystickEx
- SDL_DetachVirtualJoystick

SDL_AttachVirtualJoystickEx

Attach a new virtual joystick with extended properties.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickID SDL_AttachVirtualJoystickEx(const SDL_VirtualJoystickDesc
    *desc);
```

Function Parameters

desc	Joystick description

Return Value

Returns the joystick instance ID, or 0 if an error occurred; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AttachVirtualJoystick
- SDL_DetachVirtualJoystick

SDL_AudioDevicePaused

Use this function to query if an audio device is paused.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
SDL_bool SDL_AudioDevicePaused(SDL_AudioDeviceID dev);
```

Function Parameters

dev	a device opened by SDL_OpenAudioDevice()

Return Value

Returns SDL_TRUE if device is valid and paused, SDL_FALSE otherwise.

Remarks

Unlike in SDL2, audio devices start in an _unpaused_ state, since an app has to bind a stream before any audio will flow.

Physical devices can not be paused or unpaused, only logical devices created through SDL_OpenAudioDevice() can be. Physical and invalid device IDs will report themselves as unpaused here.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PauseAudioDevice
- SDL_ResumeAudioDevice

SDL_BindAudioStream

Bind a single audio stream to an audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_BindAudioStream(SDL_AudioDeviceID devid, SDL_AudioStream
    *stream);
```

Function Parameters

devid	an audio device to bind a stream to.
stream	an audio stream to bind to a device.

Return Value

Returns 0 on success, -1 on error; call SDL_GetError() for more information.

Remarks

This is a convenience function, equivalent to calling `SDL_BindAudioStreams(devid, &stream, 1)`.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BindAudioStreams
- SDL_UnbindAudioStream
- SDL_GetAudioStreamDevice

SDL_BindAudioStreams

Bind a list of audio streams to an audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_BindAudioStreams(SDL_AudioDeviceID devid, SDL_AudioStream
    **streams, int num_streams);
```

Function Parameters

devid	an audio device to bind a stream to.
streams	an array of audio streams to unbind.
num_streams	Number streams listed in the `streams` array.

Return Value

Returns 0 on success, -1 on error; call SDL_GetError() for more information.

Remarks

Audio data will flow through any bound streams. For an output device, data for all bound streams will be mixed together and fed to the device. For a capture device, a copy of recorded data will be provided to each bound stream.

Audio streams can only be bound to an open device. This operation is atomic–all streams bound in the same call will start processing at the same time, so they can stay in sync. Also: either all streams will be bound or none of them will be.

It is an error to bind an already-bound stream; it must be explicitly unbound first.

Binding a stream to a device will set its output format for output devices, and its input format for capture devices, so they match the device's settings. The caller is welcome to change the other end of the stream's format at any time.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BindAudioStreams
- SDL_UnbindAudioStream
- SDL_GetAudioStreamDevice

SDL_BlitSurface

Performs a fast blit from the source surface to the destination surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_BlitSurface(SDL_Surface *src, const SDL_Rect *srcrect,
    SDL_Surface *dst, SDL_Rect *dstrect);
```

Function Parameters

src	the SDL_Surface structure to be copied from
srcrect	the SDL_Rect structure representing the rectangle to be copied, or NULL to copy the entire surface
dst	the SDL_Surface structure that is the blit target
dstrect	the SDL_Rect structure representing the x and y position in the destination surface. On input the width and height are ignored (taken from srcrect), and on output this is filled in with the actual rectangle used after clipping.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This assumes that the source and destination rectangles are the same size. If either `srcrect` or `dstrect` are NULL, the entire surface (`src` or `dst`) is copied. The final blit rectangles are saved in `srcrect` and `dstrect` after all clipping is performed.

The blit function should not be called on a locked surface.

The blit semantics for surfaces with and without blending and colorkey are defined as follows:

```
RGBA->RGB:
  Source surface blend mode set to SDL_BLENDMODE_BLEND:
   alpha-blend (using the source alpha-channel and per-surface alpha)
   SDL_SRCCOLORKEY ignored.
 Source surface blend mode set to SDL_BLENDMODE_NONE:
   copy RGB.
```

```
      if SDL_SRCCOLORKEY set, only copy the pixels matching the
      RGB values of the source color key, ignoring alpha in the
      comparison.

  RGB->RGBA:
    Source surface blend mode set to SDL_BLENDMODE_BLEND:
      alpha-blend (using the source per-surface alpha)
    Source surface blend mode set to SDL_BLENDMODE_NONE:
      copy RGB, set destination alpha to source per-surface alpha value.
    both:
      if SDL_SRCCOLORKEY set, only copy the pixels matching the
      source color key.

  RGBA->RGBA:
    Source surface blend mode set to SDL_BLENDMODE_BLEND:
      alpha-blend (using the source alpha-channel and per-surface alpha)
      SDL_SRCCOLORKEY ignored.
    Source surface blend mode set to SDL_BLENDMODE_NONE:
      copy all of RGBA to the destination.
      if SDL_SRCCOLORKEY set, only copy the pixels matching the
      RGB values of the source color key, ignoring alpha in the
      comparison.

  RGB->RGB:
    Source surface blend mode set to SDL_BLENDMODE_BLEND:
      alpha-blend (using the source per-surface alpha)
    Source surface blend mode set to SDL_BLENDMODE_NONE:
      copy RGB.
    both:
      if SDL_SRCCOLORKEY set, only copy the pixels matching the
      source color key.
```

Code Examples

```
SDL_Surface *surface;
SDL_Rect source_rect;
SDL_Surface *temp_surface;

SDL_BlitSurface(surface, &source_rect, temp_surface, NULL);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BlitSurfaceScaled

SDL_BlitSurfaceScaled

Perform a scaled blit to a destination surface, which may be of a different format.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_BlitSurfaceScaled(SDL_Surface *src, const SDL_Rect *srcrect,
    SDL_Surface *dst, SDL_Rect *dstrect, SDL_ScaleMode scaleMode);
```

Function Parameters

src	the SDL_Surface structure to be copied from
srcrect	the SDL_Rect structure representing the rectangle to be copied
dst	the SDL_Surface structure that is the blit target
dstrect	the SDL_Rect structure representing the target rectangle in the destination surface, filled with the actual rectangle used after clipping
scaleMode	the SDL_ScaleMode to be used

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BlitSurface

SDL_BlitSurfaceUnchecked

Perform low-level surface blitting only.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_BlitSurfaceUnchecked(SDL_Surface *src, const SDL_Rect *srcrect,
    SDL_Surface *dst, const SDL_Rect *dstrect);
```

Function Parameters

src	the SDL_Surface structure to be copied from
srcrect	the SDL_Rect structure representing the rectangle to be copied, or NULL to copy the entire surface
dst	the SDL_Surface structure that is the blit target
dstrect	the SDL_Rect structure representing the target rectangle in the destination surface

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This is a semi-private blit function and it performs low-level surface blitting, assuming the input rectangles have already been clipped.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BlitSurface

SDL_BlitSurfaceUncheckedScaled

Perform low-level surface scaled blitting only.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_BlitSurfaceUncheckedScaled(SDL_Surface *src, const SDL_Rect
    *srcrect, SDL_Surface *dst, const SDL_Rect *dstrect, SDL_ScaleMode
    scaleMode);
```

Function Parameters

src	the SDL_Surface structure to be copied from
srcrect	the SDL_Rect structure representing the rectangle to be copied
dst	the SDL_Surface structure that is the blit target
dstrect	the SDL_Rect structure representing the target rectangle in the destination surface
scaleMode	scale algorithm to be used

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This is a semi-private function and it performs low-level surface blitting, assuming the input rectangles have already been clipped.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BlitSurfaceScaled

SDL_BroadcastCondition

Restart all threads that are waiting on the condition variable.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_BroadcastCondition(SDL_Condition *cond);
```

Function Parameters

cond	the condition variable to signal

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Code Examples

```
// BEWARE: This code example was migrated from the SDL2 Wiki, by only
    updating the names.

SDL_bool condition = SDL_FALSE;
SDL_Mutex *lock;
SDL_Condition *cond;
lock = SDL_CreateMutex();
cond = SDL_CreateCondition();

Thread_A:
    SDL_LockMutex(lock);
    while (!condition) {
        SDL_WaitCondition(cond, lock);
    }
    SDL_UnlockMutex(lock);
Thread_B:
    SDL_LockMutex(lock);
    while (!condition) {
        SDL_WaitCondition(cond, lock);
    }
    SDL_UnlockMutex(lock);
Thread_C:
    SDL_LockMutex(lock);
```

```
    /* ... */
    condition = SDL_TRUE;
    /* ... */
    SDL_BroadcastCondition(cond);
    SDL_UnlockMutex(lock);

SDL_DestroyCondition(cond);
SDL_DestroyMutex(lock);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SignalCondition
- SDL_WaitCondition
- SDL_WaitConditionTimeout

SDL_CaptureMouse

Capture the mouse and to track input outside an SDL window.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
int SDL_CaptureMouse(SDL_bool enabled);
```

Function Parameters

enabled	SDL_TRUE to enable capturing, SDL_FALSE to disable.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Capturing enables your app to obtain mouse events globally, instead of just within your window. Not all video targets support this function. When capturing is enabled, the current window will get all mouse events, but unlike relative mode, no change is made to the cursor and it is not restrained to your window.

This function may also deny mouse input to other windows–both those in your application and others on the system–so you should use this function sparingly, and in small bursts. For example, you might want to track the mouse while the user is dragging something, until the user releases a mouse button. It is not recommended that you capture the mouse for long periods of time, such as the entire time your app is running. For that, you should probably use SDL_SetRelativeMouseMode() or SDL_SetWindowMouseGrab(), depending on your goals.

While captured, mouse events still report coordinates relative to the current (foreground) window, but those coordinates may be outside the bounds of the window (including negative values). Capturing is only allowed for the foreground window. If the window loses focus while capturing, the capture will be disabled automatically.

While capturing is enabled, the current window will have the `SDL_WINDOW_MOUSE_CAPTURE` flag set.

Please note that SDL will attempt to "auto capture" the mouse while the user is pressing a button; this is to try and make mouse behavior more consistent between platforms, and deal with the common case of a user dragging the mouse

outside of the window. This means that if you are calling SDL_CaptureMouse() only to deal with this situation, you do not have to (although it is safe to do so). If this causes problems for your app, you can disable auto capture by setting the `SDL_HINT_MOUSE_AUTO_CAPTURE` hint to zero.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGlobalMouseState

SDL_ceil

Compute the ceiling of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_ceil(double x);
```

Function Parameters

x	floating point value

Return Value

Returns the ceiling of `x`

Remarks

The ceiling of `x` is the smallest integer `y` such that `y > x`, i.e `x` rounded up to the nearest integer.

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`, y integer

This function operates on double-precision floating point values, use SDL_ceilf for single-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ceilf
- SDL_floor
- SDL_trunc
- SDL_round
- SDL_lround

SDL_ceilf

Compute the ceiling of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_ceilf(float x);
```

Function Parameters

x	floating point value

Return Value

Returns the ceiling of `x`

Remarks

The ceiling of `x` is the smallest integer `y` such that `y > x`, i.e `x` rounded up to the nearest integer.

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`, y integer

This function operates on single-precision floating point values, use SDL_ceil for double-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ceil
- SDL_floorf
- SDL_truncf
- SDL_roundf
- SDL_lroundf

SDL_CleanupTLS

Cleanup all TLS data for this thread.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
void SDL_CleanupTLS(void);
```

Version

This function is available since SDL 3.0.0.

SDL_ClearAudioStream

Clear any pending data in the stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_ClearAudioStream(SDL_AudioStream *stream);
```

Function Parameters

stream	The audio stream to clear

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This drops any queued data, so there will be nothing to read from the stream until more is added.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAudioStreamAvailable
- SDL_GetAudioStreamData
- SDL_GetAudioStreamQueued
- SDL_PutAudioStreamData

SDL_ClearClipboardData

Clear the clipboard data.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
int SDL_ClearClipboardData(void);
```

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetClipboardData

SDL_ClearComposition

Dismiss the composition window/IME without disabling the subsystem.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
void SDL_ClearComposition(void);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StartTextInput
- SDL_StopTextInput

SDL_ClearError

Clear any previous error message for this thread.

Header File

Defined in SDL3/SDL_error.h

Syntax

```
void SDL_ClearError(void);
```

Code Examples

```
const char *error = SDL_GetError();
if (*error) {
  SDL_Log("SDL error: %s", error);
  SDL_ClearError();
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetError
- SDL_SetError

SDL_ClearProperty

Clear a property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_ClearProperty(SDL_PropertiesID props, const char *name);
```

Function Parameters

props	the properties to modify
name	the name of the property to clear

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_CloseAudioDevice

Close a previously-opened audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
void SDL_CloseAudioDevice(SDL_AudioDeviceID devid);
```

Function Parameters

devid	an audio device id previously returned by SDL_OpenAudioDevice()

Remarks

The application should close open audio devices once they are no longer needed.

This function may block briefly while pending audio data is played by the hardware, so that applications don't drop the last buffer of data they supplied if terminating immediately afterwards.

Code Examples

```
extern SDL_AudioSpec want;
SDL_AudioDeviceID devid =
    SDL_OpenAudioDevice(SDL_AUDIO_DEVICE_DEFAULT_OUTPUT, &want);
if (devid != 0) {
    SDL_ResumeAudioDevice(devid);
    SDL_Delay(5000); // let device play for 5 seconds
    SDL_CloseAudioDevice(devid);
}
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenAudioDevice

SDL_CloseCamera

Use this function to shut down camera processing and close the camera device.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
void SDL_CloseCamera(SDL_Camera *camera);
```

Function Parameters

camera	opened camera device

Thread Safety

It is safe to call this function from any thread, but no thread may reference `device` once this function is called.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenCameraWithSpec
- SDL_OpenCamera

SDL_CloseGamepad

Close a gamepad previously opened with SDL_OpenGamepad().

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
void SDL_CloseGamepad(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	a gamepad identifier previously returned by SDL_OpenGamepad()

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenGamepad

SDL_CloseHaptic

Close a haptic device previously opened with SDL_OpenHaptic().

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
void SDL_CloseHaptic(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic device to close

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenHaptic

SDL_CloseIO

Close and free an allocated SDL_IOStream structure.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
int SDL_CloseIO(SDL_IOStream *context);
```

Function Parameters

context	SDL_IOStream structure to close

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

SDL_CloseIO() closes and cleans up the SDL_IOStream stream. It releases any resources used by the stream and frees the SDL_IOStream itself. This returns 0 on success, or -1 if the stream failed to flush to its output (e.g. to disk).

Note that if this fails to flush the stream to disk, this function reports an error, but the SDL_IOStream is still invalid once this function returns.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenIO

SDL_CloseJoystick

Close a joystick previously opened with SDL_OpenJoystick().

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
void SDL_CloseJoystick(SDL_Joystick *joystick);
```

Function Parameters

joystick	The joystick device to close

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenJoystick

SDL_CloseSensor

Close a sensor previously opened with SDL_OpenSensor().

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
void SDL_CloseSensor(SDL_Sensor *sensor);
```

Function Parameters

sensor	The SDL_Sensor object to close

Version

This function is available since SDL 3.0.0.

SDL_CloseStorage

Closes and frees a storage container.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
int SDL_CloseStorage(SDL_Storage *storage);
```

Function Parameters

storage	a storage container to close

Return Value

Returns 0 if the container was freed with no errors, a negative value otherwise; call SDL_GetError() for more information. Even if the function returns an error, the container data will be freed; the error is only for informational purposes.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenFileStorage
- SDL_OpenStorage
- SDL_OpenTitleStorage
- SDL_OpenUserStorage

SDL_ComposeCustomBlendMode

Compose a custom blend mode for renderers.

Header File

Defined in SDL3/SDL_blendmode.h

Syntax

```
SDL_BlendMode SDL_ComposeCustomBlendMode(SDL_BlendFactor srcColorFactor,
                                         SDL_BlendFactor dstColorFactor,
                                         SDL_BlendOperation colorOperation,
                                         SDL_BlendFactor srcAlphaFactor,
                                         SDL_BlendFactor dstAlphaFactor,
                                         SDL_BlendOperation alphaOperation);
```

Function Parameters

srcColorFactor	the SDL_BlendFactor applied to the red, green, and blue components of the source pixels
dstColorFactor	the SDL_BlendFactor applied to the red, green, and blue components of the destination pixels
colorOperation	the SDL_BlendOperation used to combine the red, green, and blue components of the source and destination pixels
srcAlphaFactor	the SDL_BlendFactor applied to the alpha component of the source pixels
dstAlphaFactor	the SDL_BlendFactor applied to the alpha component of the destination pixels
alphaOperation	the SDL_BlendOperation used to combine the alpha component of the source and destination pixels

Return Value

Returns an SDL_BlendMode that represents the chosen factors and operations.

Remarks

The functions SDL_SetRenderDrawBlendMode and SDL_SetTextureBlendMode accept the SDL_BlendMode returned by this function if the renderer supports it.

A blend mode controls how the pixels from a drawing operation (source) get combined with the pixels from the render target (destination). First, the components of the source and destination pixels get multiplied with their blend

factors. Then, the blend operation takes the two products and calculates the result that will get stored in the render target.

Expressed in pseudocode, it would look like this:

```
dstRGB = colorOperation(srcRGB * srcColorFactor, dstRGB *
    dstColorFactor);
dstA = alphaOperation(srcA * srcAlphaFactor, dstA * dstAlphaFactor);
```

Where the functions `colorOperation(src, dst)` and `alphaOperation(src, dst)` can return one of the following:

- `src + dst`
- `src - dst`
- `dst - src`
- `min(src, dst)`
- `max(src, dst)`

The red, green, and blue components are always multiplied with the first, second, and third components of the SDL_BlendFactor, respectively. The fourth component is not used.

The alpha component is always multiplied with the fourth component of the SDL_BlendFactor. The other components are not used in the alpha calculation.

Support for these blend modes varies for each renderer. To check if a specific SDL_BlendMode is supported, create a renderer and pass it to either SDL_SetRenderDrawBlendMode or SDL_SetTextureBlendMode. They will return with an error if the blend mode is not supported.

This list describes the support of custom blend modes for each renderer. All renderers support the four blend modes listed in the SDL_BlendMode enumeration.

- **direct3d**: Supports all operations with all factors. However, some factors produce unexpected results with `SDL_BLENDOPERATION_MINIMUM` and `SDL_BLENDOPERATION_MAXIMUM`.
- **direct3d11**: Same as Direct3D 9.
- **opengl**: Supports the `SDL_BLENDOPERATION_ADD` operation with all factors. OpenGL versions 1.1, 1.2, and 1.3 do not work correctly here.
- **opengles2**: Supports the `SDL_BLENDOPERATION_ADD`, `SDL_BLENDOPERATION_SUBTRACT`, `SDL_BLENDOPERATION_REV_SUBTRACT` operations with all factors.
- **psp**: No custom blend mode support.
- **software**: No custom blend mode support.

Some renderers do not provide an alpha component for the default render target. The `SDL_BLENDFACTOR_DST_ALPHA` and `SDL_BLENDFACTOR_ONE_MINUS_DST_ALPHA` factors do not have an effect in this case.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderDrawBlendMode
- SDL_GetRenderDrawBlendMode
- SDL_SetTextureBlendMode
- SDL_GetTextureBlendMode

SDL_ConvertAudioSamples

Convert some audio data of one format to another format.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_ConvertAudioSamples(const SDL_AudioSpec *src_spec,
                            const Uint8 *src_data,
                            int src_len,
                            const SDL_AudioSpec *dst_spec,
                            Uint8 **dst_data,
                            int *dst_len);
```

Function Parameters

src_spec	The format details of the input audio
src_data	The audio data to be converted
src_len	The len of src_data
dst_spec	The format details of the output audio
dst_data	Will be filled with a pointer to converted audio data, which should be freed with SDL_free(). On error, it will be NULL.
dst_len	Will be filled with the len of dst_data

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Please note that this function is for convenience, but should not be used to resample audio in blocks, as it will introduce audio artifacts on the boundaries. You should only use this function if you are converting audio data in its entirety in one call. If you want to convert audio in smaller chunks, use an SDL_AudioStream, which is designed for this situation.

Internally, this function creates and destroys an SDL_AudioStream on each use, so it's also less efficient than using one directly, if you need to convert multiple times.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_ConvertEventToRenderCoordinates

Convert the coordinates in an event to render coordinates.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_ConvertEventToRenderCoordinates(SDL_Renderer *renderer,
    SDL_Event *event);
```

Function Parameters

renderer	the rendering context
event	the event to modify

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Touch coordinates are converted from normalized coordinates in the window to non-normalized rendering coordinates.

Once converted, the coordinates may be outside the rendering area.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderCoordinatesFromWindowCoordinates

SDL_ConvertPixels

Copy a block of pixels of one format to another format.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_ConvertPixels(int width, int height, SDL_PixelFormatEnum
    src_format, const void *src, int src_pitch, SDL_PixelFormatEnum
    dst_format, void *dst, int dst_pitch);
```

Function Parameters

width	the width of the block to copy, in pixels
height	the height of the block to copy, in pixels
src_format	an SDL_PixelFormatEnum value of the `src` pixels format
src	a pointer to the source pixels
src_pitch	the pitch of the source pixels, in bytes
dst_format	an SDL_PixelFormatEnum value of the `dst` pixels format
dst	a pointer to be filled in with new pixel data
dst_pitch	the pitch of the destination pixels, in bytes

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ConvertPixelsAndColorspace

SDL_ConvertPixelsAndColorspace

Copy a block of pixels of one format and colorspace to another format and colorspace.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_ConvertPixelsAndColorspace(int width, int height,
    SDL_PixelFormatEnum src_format, SDL_Colorspace src_colorspace,
    SDL_PropertiesID src_properties, const void *src, int src_pitch,
    SDL_PixelFormatEnum dst_format, SDL_Colorspace dst_colorspace,
    SDL_PropertiesID dst_properties, void *dst, int dst_pitch);
```

Function Parameters

width	the width of the block to copy, in pixels
height	the height of the block to copy, in pixels
src_format	an SDL_PixelFormatEnum value of the `src` pixels format
src_colorspace	an SDL_ColorSpace value describing the colorspace of the `src` pixels
src_properties	an SDL_PropertiesID with additional source color properties, or 0
src	a pointer to the source pixels
src_pitch	the pitch of the source pixels, in bytes
dst_format	an SDL_PixelFormatEnum value of the `dst` pixels format
dst_colorspace	an SDL_ColorSpace value describing the colorspace of the `dst` pixels
dst_properties	an SDL_PropertiesID with additional destination color properties, or 0
dst	a pointer to be filled in with new pixel data
dst_pitch	the pitch of the destination pixels, in bytes

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ConvertPixels

SDL_ConvertSurface

Copy an existing surface to a new surface of the specified format.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_Surface* SDL_ConvertSurface(SDL_Surface *surface, const
    SDL_PixelFormat *format);
```

Function Parameters

surface	the existing SDL_Surface structure to convert
format	the SDL_PixelFormat structure that the new surface is optimized for

Return Value

Returns the new SDL_Surface structure that is created or NULL if it fails; call SDL_GetError() for more information.

Remarks

This function is used to optimize images for faster *repeat* blitting. This is accomplished by converting the original and storing the result as a new surface. The new, optimized surface can then be used as the source for future blits, making them faster.

Code Examples

```
SDL_Surface *input;

SDL_PixelFormat *format = SDL_CreatePixelFormat(SDL_PIXELFORMAT_RGBA32);
SDL_Surface *output = SDL_ConvertSurface(input, format);
SDL_DestroyPixelFormat(format);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ConvertSurfaceFormat
- SDL_ConvertSurfaceFormatAndColorspace
- SDL_CreatePixelFormat
- SDL_DestroySurface

SDL_ConvertSurfaceFormat

Copy an existing surface to a new surface of the specified format.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_Surface* SDL_ConvertSurfaceFormat(SDL_Surface *surface,
    SDL_PixelFormatEnum pixel_format);
```

Function Parameters

surface	the existing SDL_Surface structure to convert
pixel_format	the new pixel format

Return Value

Returns the new SDL_Surface structure that is created or NULL if it fails; call SDL_GetError() for more information.

Remarks

This function operates just like SDL_ConvertSurface(), but accepts an SDL_PixelFormatEnum value instead of an SDL_PixelFormat structure. As such, it might be easier to call but it doesn't have access to palette information for the destination surface, in case that would be important.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ConvertSurface
- SDL_ConvertSurfaceFormatAndColorspace
- SDL_DestroySurface

SDL_ConvertSurfaceFormatAndColorspace

Copy an existing surface to a new surface of the specified format and colorspace.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_Surface* SDL_ConvertSurfaceFormatAndColorspace(SDL_Surface *surface,
    SDL_PixelFormatEnum pixel_format, SDL_Colorspace colorspace,
    SDL_PropertiesID props);
```

Function Parameters

surface	the existing SDL_Surface structure to convert
pixel_format	the new pixel format
colorspace	the new colorspace
props	an SDL_PropertiesID with additional color properties, or 0

Return Value

Returns the new SDL_Surface structure that is created or NULL if it fails; call SDL_GetError() for more information.

Remarks

This function converts an existing surface to a new format and colorspace and returns the new surface. This will perform any pixel format and colorspace conversion needed.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ConvertSurface
- SDL_ConvertSurfaceFormat
- SDL_DestroySurface

SDL_CopyProperties

Copy a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_CopyProperties(SDL_PropertiesID src, SDL_PropertiesID dst);
```

Function Parameters

src	the properties to copy
dst	the destination properties

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Copy all the properties from one set of properties to another, with the exception of properties requiring cleanup (set using SDL_SetPropertyWithCleanup()), which will not be copied. Any property that already exists on `dst` will be overwritten.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_copysign

Copy the sign of one floating-point value to another.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_copysign(double x, double y);
```

Function Parameters

x	floating point value to use as the magnitude
y	floating point value to use as the sign

Return Value

Returns the floating point value with the sign of y and the magnitude of x

Remarks

The definition of copysign is that copysign(x, y) = abs(x) * sign(y) .

Domain: `-INF <= x <= INF`, -INF ¡= y ¡= f

Range: `-INF <= z <= INF`

This function operates on double-precision floating point values, use SDL_copysignf for single-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_copysignf
- SDL_fabs

SDL_copysignf

Copy the sign of one floating-point value to another.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_copysignf(float x, float y);
```

Function Parameters

x	floating point value to use as the magnitude
y	floating point value to use as the sign

Return Value

Returns the floating point value with the sign of y and the magnitude of x

Remarks

The definition of copysign is that copysign(x, y) = abs(x) * sign(y) .

Domain: `-INF <= x <= INF`, -INF ¡= y ¡= f

Range: `-INF <= z <= INF`

This function operates on single-precision floating point values, use SDL_copysign for double-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_copysignf
- SDL_fabsf

SDL_cos

Compute the cosine of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_cos(double x);
```

Function Parameters

x	floating point value, in radians

Return Value

Returns cosine of `x`

Remarks

Domain: `-INF <= x <= INF`

Range: `-1 <= y <= 1`

This function operates on double-precision floating point values, use SDL_cosf for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_cosf
- SDL_acos
- SDL_sin

SDL_cosf

Compute the cosine of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_cosf(float x);
```

Function Parameters

x	floating point value, in radians

Return Value

Returns cosine of `x`

Remarks

Domain: `-INF <= x <= INF`

Range: `-1 <= y <= 1`

This function operates on single-precision floating point values, use SDL_cos for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_cos
- SDL_acosf
- SDL_sinf

SDL_CreateAudioStream

Create a new audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
SDL_AudioStream* SDL_CreateAudioStream(const SDL_AudioSpec *src_spec,
    const SDL_AudioSpec *dst_spec);
```

Function Parameters

src_spec	The format details of the input audio
dst_spec	The format details of the output audio

Return Value

Returns a new audio stream on success, or NULL on failure.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PutAudioStreamData
- SDL_GetAudioStreamData
- SDL_GetAudioStreamAvailable
- SDL_FlushAudioStream
- SDL_ClearAudioStream
- SDL_ChangeAudioStreamOutput
- SDL_DestroyAudioStream

SDL_CreateColorCursor

Create a color cursor.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_Cursor* SDL_CreateColorCursor(SDL_Surface *surface,
                                  int hot_x,
                                  int hot_y);
```

Function Parameters

surface	an SDL_Surface structure representing the cursor image
hot_x	the x position of the cursor hot spot
hot_y	the y position of the cursor hot spot

Return Value

Returns the new cursor on success or NULL on failure; call SDL_GetError() for more information.

Code Examples

```
#include <SDL3/SDL.h>

int
main(int argc, char *argv[])
{
    SDL_Window *window = NULL;
    SDL_Renderer *renderer = NULL;
    SDL_Surface *surface = NULL;
    SDL_Cursor *cursor = NULL;
    SDL_bool error = SDL_TRUE;

    if (SDL_Init(SDL_INIT_VIDEO) < 0) {
        goto exit;
    }
    if (SDL_CreateWindowAndRenderer("Hello SDL", 640, 480, 0, &window,
         &renderer) < 0) {
        goto exit;
    }
    surface = SDL_LoadBMP((1 < argc) ? argv[1] : "cursor.bmp");
```

```
    if (!surface) {
        goto exit;
    }
    cursor = SDL_CreateColorCursor(surface, 0, 0);
    if (!cursor) {
        goto exit;
    }

    SDL_SetCursor(cursor);
    SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);
    while (SDL_TRUE) {
        SDL_Event event;
        while (SDL_PollEvent(&event)) {
            switch (event.type) {
            case SDL_EVENT_MOUSE_BUTTON_UP:
            case SDL_EVENT_QUIT:
                error = SDL_FALSE;
                goto exit;
            }
        }
        SDL_RenderClear(renderer);
        SDL_RenderPresent(renderer);
    }

exit:
    if (error) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "%s", SDL_GetError());
    }
    if (cursor) {
        SDL_DestroyCursor(cursor);
    }
    if (surface) {
        SDL_DestroySurface(surface);
    }
    if (renderer) {
        SDL_DestroyRenderer(renderer);
    }
    if (window) {
        SDL_DestroyWindow(window);
    }
    SDL_Quit();
    return error;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateCursor
- SDL_CreateSystemCursor
- SDL_DestroyCursor
- SDL_SetCursor

SDL_CreateCondition

Create a condition variable.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
SDL_Condition* SDL_CreateCondition(void);
```

Return Value

Returns a new condition variable or NULL on failure; call SDL_GetError() for more information.

Code Examples

Typical use of condition variables:

```
// BEWARE: This code example was migrated from the SDL2 Wiki, by only
    updating the names.

SDL_bool condition = SDL_FALSE;
SDL_Mutex *lock;
SDL_Condition *cond;
lock = SDL_CreateMutex();
cond = SDL_CreateCondition();


Thread_A:
    SDL_LockMutex(lock);
    while (!condition) {
        SDL_WaitCondition(cond, lock);
    }
    SDL_UnlockMutex(lock);
Thread_B:
    SDL_LockMutex(lock);
    /* ... */
    condition = SDL_TRUE;
    /* ... */
    SDL_SignalCondition(cond);
    SDL_UnlockMutex(lock);

SDL_DestroyCondition(cond);
SDL_DestroyMutex(lock);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BroadcastCondition
- SDL_SignalCondition
- SDL_WaitCondition
- SDL_WaitConditionTimeout
- SDL_DestroyCondition

SDL_CreateCursor

Create a cursor using the specified bitmap data and mask (in MSB format).

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_Cursor* SDL_CreateCursor(const Uint8 * data,
                             const Uint8 * mask,
                             int w, int h, int hot_x,
                             int hot_y);
```

Function Parameters

data	the color value for each pixel of the cursor
mask	the mask value for each pixel of the cursor
w	the width of the cursor
h	the height of the cursor
hot_x	the X-axis location of the upper left corner of the cursor relative to the actual mouse position
hot_y	the Y-axis location of the upper left corner of the cursor relative to the actual mouse position

Return Value

Returns a new cursor with the specified parameters on success or NULL on failure; call SDL_GetError() for more information.

Remarks

`mask` has to be in MSB (Most Significant Bit) format.

The cursor width (`w`) must be a multiple of 8 bits.

The cursor is created in black and white according to the following:

- data=0, mask=1: white
- data=1, mask=1: black
- data=0, mask=0: transparent
- data=1, mask=0: inverted color if possible, black if not.

Cursors created with this function must be freed with SDL_DestroyCursor().

If you want to have a color cursor, or create your cursor from an SDL_Surface, you should use SDL_CreateColorCursor(). Alternately, you can hide the cursor and draw your own as part of your game's rendering, but it will be bound to the framerate.

Also, SDL_CreateSystemCursor() is available, which provides several readily-available system cursors to pick from.

Code Examples

```
/* Stolen from the mailing list */
/* Creates a new mouse cursor from an XPM */


/* XPM */
static const char *arrow[] = {
  /* width height num_colors chars_per_pixel */
  "    32    32       3           1",
  /* colors */
  "X c #000000",
  ". c #ffffff",
  "  c None",
  /* pixels */
  "X                               ",
  "XX                              ",
  "X.X                             ",
  "X..X                            ",
  "X...X                           ",
  "X....X                          ",
  "X.....X                         ",
  "X......X                        ",
  "X.......X                       ",
  "X........X                      ",
  "X.....XXXXX                     ",
  "X..X..X                         ",
  "X.X X..X                        ",
  "XX  X..X                        ",
  "X    X..X                       ",
  "     X..X                       ",
  "      X..X                      ",
  "      X..X                      ",
  "       XX                       ",
  "                                ",
  "                                ",
  "                                ",
  "                                ",
  "                                ",
  "                                ",
```

```
  "                               ",
  "                               ",
  "                               ",
  "                               ",
  "                               ",
  "                               ",
  "                               ",
  "0,0"
};

SDL_Cursor *init_system_cursor(const char *image[])
{
  int i, row, col;
  Uint8 data[4*32];
  Uint8 mask[4*32];
  int hot_x, hot_y;

  i = -1;
  for (row=0; row<32; ++row) {
    for (col=0; col<32; ++col) {
      if (col % 8) {
        data[i] <<= 1;
        mask[i] <<= 1;
      } else {
        ++i;
        data[i] = mask[i] = 0;
      }
      switch (image[4+row][col]) {
        case 'X':
          data[i] |= 0x01;
          mask[i] |= 0x01;
          break;
        case '.':
          mask[i] |= 0x01;
          break;
        case ' ':
          break;
      }
    }
  }
  sscanf(image[4+row], "%d,%d", &hot_x, &hot_y);
  return SDL_CreateCursor(data, mask, 32, 32, hot_x, hot_y);
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateColorCursor
- SDL_CreateSystemCursor
- SDL_DestroyCursor
- SDL_SetCursor

SDL_CreateDirectory

Create a directory.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
int SDL_CreateDirectory(const char *path);
```

Function Parameters

path	the path of the directory to create

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_CreateHapticEffect

Create a new haptic effect on a specified device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_CreateHapticEffect(SDL_Haptic *haptic, const SDL_HapticEffect
    *effect);
```

Function Parameters

haptic	an SDL_Haptic device to create the effect on
effect	an SDL_HapticEffect structure containing the properties of the effect to create

Return Value

Returns the ID of the effect on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroyHapticEffect
- SDL_RunHapticEffect
- SDL_UpdateHapticEffect

SDL_CreateMutex

Create a new mutex.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
SDL_Mutex* SDL_CreateMutex(void);
```

Return Value

Returns the initialized and unlocked mutex or NULL on failure; call SDL_GetError() for more information.

Remarks

All newly-created mutexes begin in the _unlocked_ state.

Calls to SDL_LockMutex() will not return while the mutex is locked by another thread. See SDL_TryLockMutex() to attempt to lock without blocking.

SDL mutexes are reentrant.

Code Examples

¡!– # Begin Mutex Example –¿

```
SDL_Mutex *mutex;

mutex = SDL_CreateMutex();
if (!mutex) {
  SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create mutex\n");
  return 1;
}

if (SDL_TryLockMutex(mutex) == 0) {
  /* Do stuff while mutex is locked */
  SDL_UnlockMutex(mutex);
} else {
  SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't lock mutex\n");
}

SDL_DestroyMutex(mutex);
```

¡!– # End Mutex Example –¿

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroyMutex
- SDL_LockMutex
- SDL_TryLockMutex
- SDL_UnlockMutex

SDL_CreatePalette

Create a palette structure with the specified number of color entries.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
SDL_Palette* SDL_CreatePalette(int ncolors);
```

Function Parameters

ncolors	represents the number of color entries in the color palette

Return Value

Returns a new SDL_Palette structure on success or NULL on failure (e.g. if there wasn't enough memory); call SDL_GetError() for more information.

Remarks

The palette entries are initialized to white.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroyPalette
- SDL_SetPaletteColors
- SDL_SetPixelFormatPalette

SDL_CreatePixelFormat

Create an SDL_PixelFormat structure corresponding to a pixel format.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
SDL_PixelFormat * SDL_CreatePixelFormat(SDL_PixelFormatEnum
    pixel_format);
```

Function Parameters

pixel_format	one of the SDL_PixelFormatEnum values

Return Value

Returns the new SDL_PixelFormat structure or NULL on failure; call SDL_GetError() for more information.

Remarks

Returned structure may come from a shared global cache (i.e. not newly allocated), and hence should not be modified, especially the palette. Weird errors such as `Blit combination not supported` may occur.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroyPixelFormat
- SDL_SetPixelFormatPalette

SDL_CreatePopupWindow

Create a child popup window of the specified parent window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_Window* SDL_CreatePopupWindow(SDL_Window *parent, int offset_x, int
    offset_y, int w, int h, SDL_WindowFlags flags);
```

Function Parameters

parent	the parent of the window, must not be NULL
offset_x	the x position of the popup window relative to the origin of the parent
offset_y	the y position of the popup window relative to the origin of the parent window
w	the width of the window
h	the height of the window
flags	SDL_WINDOW_TOOLTIP or SDL_WINDOW_POPUP_MENU, and zero or more additional SDL_WindowFlags OR'd together.

Return Value

Returns the window that was created or NULL on failure; call SDL_GetError() for more information.

Remarks

'flags' **must** contain exactly one of the following: - 'SDL_WINDOW_TOOLTIP': The popup window is a tooltip and will not pass any input events. - 'SDL_WINDOW_POPUP_MENU': The popup window is a popup menu. The topmost popup menu will implicitly gain the keyboard focus.

The following flags are not relevant to popup window creation and will be ignored:

- 'SDL_WINDOW_MINIMIZED'
- 'SDL_WINDOW_MAXIMIZED'
- 'SDL_WINDOW_FULLSCREEN'

- 'SDL_WINDOW_BORDERLESS'

The parent parameter **must** be non-null and a valid window. The parent of a popup window can be either a regular, toplevel window, or another popup window.

Popup windows cannot be minimized, maximized, made fullscreen, raised, flash, be made a modal window, be the parent of a modal window, or grab the mouse and/or keyboard. Attempts to do so will fail.

Popup windows implicitly do not have a border/decorations and do not appear on the taskbar/dock or in lists of windows such as alt-tab menus.

If a parent window is hidden, any child popup windows will be recursively hidden as well. Child popup windows not explicitly hidden will be restored when the parent is shown.

If the parent window is destroyed, any child popup windows will be recursively destroyed as well.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateWindow
- SDL_CreateWindowWithProperties
- SDL_DestroyWindow
- SDL_GetWindowParent

SDL_CreateProperties

Create a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
SDL_PropertiesID SDL_CreateProperties(void);
```

Return Value

Returns an ID for a new set of properties, or 0 on failure; call SDL_GetError() for more information.

Remarks

All properties are automatically destroyed when SDL_Quit() is called.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroyProperties

SDL_CreateRenderer

Create a 2D rendering context for a window.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Renderer * SDL_CreateRenderer(SDL_Window *window, const char *name,
    Uint32 flags);
```

Function Parameters

window	the window where rendering is displayed
name	the name of the rendering driver to initialize, or NULL to initialize the first one supporting the requested flags
flags	0, or one or more SDL_RendererFlags OR'd together

Return Value

Returns a valid rendering context or NULL if there was an error; call SDL_GetError() for more information.

Remarks

If you want a specific renderer, you can specify its name here. A list of available renderers can be obtained by calling SDL_GetRenderDriver multiple times, with indices from 0 to SDL_GetNumRenderDrivers()-1. If you don't need a specific renderer, specify NULL and SDL will attempt to choose the best option for you, based on what is available on the user's system.

By default the rendering size matches the window size in pixels, but you can call SDL_SetRenderLogicalPresentation() to change the content size and scaling options.

Code Examples

```
#include <SDL3/SDL.h>
#include <SDL3/SDL_main.h>

int main(int argc, char *argv[])
{
    SDL_Window *win = NULL;
    SDL_Renderer *renderer = NULL;
```

```
    SDL_Texture *bitmapTex = NULL;
    SDL_Surface *bitmapSurface = NULL;
    int width = 320, height = 240;
    SDL_bool loopShouldStop = SDL_FALSE;

    SDL_Init(SDL_INIT_VIDEO);

    win = SDL_CreateWindow("Hello World", width, height, 0);

    renderer = SDL_CreateRenderer(win, NULL, 0);

    bitmapSurface = SDL_LoadBMP("img/hello.bmp");
    bitmapTex = SDL_CreateTextureFromSurface(renderer, bitmapSurface);
    SDL_DestroySurface(bitmapSurface);

    while (!loopShouldStop)
    {
        SDL_Event event;
        while (SDL_PollEvent(&event))
        {
            switch (event.type)
            {
                case SDL_EVENT_QUIT:
                    loopShouldStop = SDL_TRUE;
                    break;
            }
        }

        SDL_RenderClear(renderer);
        SDL_RenderTexture(renderer, bitmapTex, NULL, NULL);
        SDL_RenderPresent(renderer);
    }

    SDL_DestroyTexture(bitmapTex);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(win);

    SDL_Quit();

    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateRendererWithProperties
- SDL_CreateSoftwareRenderer
- SDL_DestroyRenderer
- SDL_GetNumRenderDrivers
- SDL_GetRenderDriver
- SDL_GetRendererInfo

SDL_CreateRendererWithProperties

Create a 2D rendering context for a window, with the specified properties.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Renderer * SDL_CreateRendererWithProperties(SDL_PropertiesID props);
```

Function Parameters

props	the properties to use

Return Value

Returns a valid rendering context or NULL if there was an error; call SDL_GetError() for more information.

Remarks

These are the supported properties:

- `SDL_PROP_RENDERER_CREATE_NAME_STRING`: the name of the rendering driver to use, if a specific one is desired
- `SDL_PROP_RENDERER_CREATE_WINDOW_POINTER`: the window where rendering is displayed, required if this isn't a software renderer using a surface
- `SDL_PROP_RENDERER_CREATE_SURFACE_POINTER`: the surface where rendering is displayed, if you want a software renderer without a window
- `SDL_PROP_RENDERER_CREATE_OUTPUT_COLORSPACE_NUMBER`: an SDL_ColorSpace value describing the colorspace for output to the display, defaults to SDL_COLORSPACE_SRGB. The direct3d11, direct3d12, and metal renderers support SDL_COLORSPACE_SRGB_LINEAR, which is a linear color space and supports HDR output. If you select SDL_COLORSPACE_SRGB_LINEAR, drawing still uses the sRGB colorspace, but values can go beyond 1.0 and float (linear) format textures can be used for HDR content.
- `SDL_PROP_RENDERER_CREATE_PRESENT_VSYNC_BOOLEAN`: true if you want present synchronized with the refresh rate

With the vulkan renderer:

- `SDL_PROP_RENDERER_CREATE_VULKAN_INSTANCE_POINTER`: the VkInstance to use with the renderer, optional.
- `SDL_PROP_RENDERER_CREATE_VULKAN_SURFACE_NUMBER`: the VkSurfaceKHR to use with the renderer, optional.
- `SDL_PROP_RENDERER_CREATE_VULKAN_PHYSICAL_DEVICE_POINTER`: the VkPhysicalDevice to use with the renderer, optional.
- `SDL_PROP_RENDERER_CREATE_VULKAN_DEVICE_POINTER`: the VkDevice to use with the renderer, optional.
- `SDL_PROP_RENDERER_CREATE_VULKAN_GRAPHICS_QUEUE_FAMILY_INDEX_NUMBER`: the queue family index used for rendering.
- `SDL_PROP_RENDERER_CREATE_VULKAN_PRESENT_QUEUE_FAMILY_INDEX_NUMBER`: the queue family index used for presentation.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateProperties
- SDL_CreateRenderer
- SDL_CreateSoftwareRenderer
- SDL_DestroyRenderer
- SDL_GetRendererInfo

SDL_CreateRWLock

Create a new read/write lock.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
SDL_RWLock* SDL_CreateRWLock(void);
```

Return Value

Returns the initialized and unlocked read/write lock or NULL on failure; call SDL_GetError() for more information.

Remarks

A read/write lock is useful for situations where you have multiple threads trying to access a resource that is rarely updated. All threads requesting a read-only lock will be allowed to run in parallel; if a thread requests a write lock, it will be provided exclusive access. This makes it safe for multiple threads to use a resource at the same time if they promise not to change it, and when it has to be changed, the rwlock will serve as a gateway to make sure those changes can be made safely.

In the right situation, a rwlock can be more efficient than a mutex, which only lets a single thread proceed at a time, even if it won't be modifying the data.

All newly-created read/write locks begin in the _unlocked_ state.

Calls to SDL_LockRWLockForReading() and SDL_LockRWLockForWriting will not return while the rwlock is locked _for writing_ by another thread. See SDL_TryLockRWLockForReading() and SDL_TryLockRWLockForWriting() to attempt to lock without blocking.

SDL read/write locks are only recursive for read-only locks! They are not guaranteed to be fair, or provide access in a FIFO manner! They are not guaranteed to favor writers. You may not lock a rwlock for both read-only and write access at the same time from the same thread (so you can't promote your read-only lock to a write lock without unlocking first).

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroyRWLock
- SDL_LockRWLockForReading
- SDL_LockRWLockForWriting
- SDL_TryLockRWLockForReading
- SDL_TryLockRWLockForWriting
- SDL_UnlockRWLock

SDL_CreateSemaphore

Create a semaphore.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
SDL_Semaphore* SDL_CreateSemaphore(Uint32 initial_value);
```

Function Parameters

initial_value	the starting value of the semaphore

Return Value

Returns a new semaphore or NULL on failure; call SDL_GetError() for more information.

Remarks

This function creates a new semaphore and initializes it with the value `initial_value`. Each wait operation on the semaphore will atomically decrement the semaphore value and potentially block if the semaphore value is 0. Each post operation will atomically increment the semaphore value and wake waiting threads and allow them to retry the wait operation.

Code Examples

¡!– # Begin Semaphore Example –¿ Typical use of semaphores:

```
void add_data_to_queue(void);
void get_data_to_queue(void);
void get_data_from_queue(void);
int data_available(void);
void wait_for_threads(void);

SDL_AtomicInt done;
SDL_Semaphore *sem;

SDL_AtomicSet(&done, 0);
sem = SDL_CreateSemaphore(0);
```

```
Thread_A:
    while (!SDL_AtomicGet(&done)) {
        add_data_to_queue();
        SDL_PostSemaphore(sem);
    }

Thread_B:
    while (!SDL_AtomicGet(&done)) {
        SDL_WaitSemaphore(sem);
        if (data_available()) {
            get_data_from_queue();
        }
    }


SDL_AtomicSet(&done, 1);
SDL_PostSemaphore(sem);
wait_for_threads();
SDL_DestroySemaphore(sem);
```

¡!– # End Semaphore Example –¿

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroySemaphore
- SDL_PostSemaphore
- SDL_TryWaitSemaphore
- SDL_GetSemaphoreValue
- SDL_WaitSemaphore
- SDL_WaitSemaphoreTimeout

SDL_CreateSoftwareRenderer

Create a 2D software rendering context for a surface.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Renderer* SDL_CreateSoftwareRenderer(SDL_Surface *surface);
```

Function Parameters

surface	the SDL_Surface structure representing the surface where rendering is done

Return Value

Returns a valid rendering context or NULL if there was an error; call SDL_GetError() for more information.

Remarks

Two other API which can be used to create SDL_Renderer: SDL_CreateRenderer() and SDL_CreateWindowAndRenderer(). These can _also_ create a software renderer, but they are intended to be used with an SDL_Window as the final destination and not an SDL_Surface.

Code Examples

```
#include <SDL3/SDL.h>
#include <SDL3/SDL_main.h>

SDL_Window *window;
SDL_Renderer *renderer;
int done;

void DrawChessBoard(SDL_Renderer *renderer)
{
    int row = 0, column = 0, x = 0;
    SDL_FRect rect;
    SDL_Rect darea;

    /* Get the Size of drawing surface */
    SDL_GetRenderViewport(renderer, &darea);
```

```
    SDL_SetRenderDrawColor(renderer, 0xFF, 0xFF, 0xFF, 0xFF);
    SDL_RenderClear(renderer);

    for (; row < 8; row++) {
        column = row % 2;
        x = column;
        for (; column < 4 + (row % 2); column++) {
            SDL_SetRenderDrawColor(renderer, 0, 0, 0, 0xFF);

            rect.w = (float)darea.w / 8;
            rect.h = (float)darea.h / 8;
            rect.x = x * rect.w;
            rect.y = row * rect.h;
            x = x + 2;
            SDL_RenderFillRect(renderer, &rect);
        }
    }
    SDL_RenderPresent(renderer);
}

void loop()
{
    SDL_Event e;
    while (SDL_PollEvent(&e)) {
        if (e.type == SDL_EVENT_QUIT) {
            done = 1;
            return;
        }

        if ((e.type == SDL_EVENT_KEY_DOWN) && (e.key.keysym.sym ==
             SDLK_ESCAPE)) {
            done = 1;
            return;
        }
    }

    DrawChessBoard(renderer);

    /* Got everything on rendering surface,
       now Update the drawing image on window screen */
    SDL_UpdateWindowSurface(window);
}

int main(int argc, char *argv[])
{
    SDL_Surface *surface;

    /* Enable standard application logging */
```

```
    SDL_LogSetPriority(SDL_LOG_CATEGORY_APPLICATION,
        SDL_LOG_PRIORITY_INFO);

    /* Initialize SDL */
    if (SDL_Init(SDL_INIT_VIDEO) != 0) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "SDL_Init fail :
            %s\n", SDL_GetError());
        return 1;
    }

    /* Create window and renderer for given surface */
    window = SDL_CreateWindow("Chess Board", 640, 480, 0);
    if (!window) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Window creation fail
            : %s\n", SDL_GetError());
        return 1;
    }
    surface = SDL_GetWindowSurface(window);
    renderer = SDL_CreateSoftwareRenderer(surface);
    if (!renderer) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Render creation for
            surface fail : %s\n", SDL_GetError());
        return 1;
    }

    /* Draw the Image on rendering surface */
    done = 0;

    while (!done) {
        loop();
    }

    SDL_Quit();
    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroyRenderer

SDL_CreateStorageDirectory

Create a directory in a writable storage container.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
int SDL_CreateStorageDirectory(SDL_Storage *storage, const char *path);
```

Function Parameters

storage	a storage container
path	the path of the directory to create

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StorageReady

SDL_CreateSurface

Allocate a new RGB surface with a specific pixel format.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_Surface* SDL_CreateSurface(int width, int height,
    SDL_PixelFormatEnum format);
```

Function Parameters

width	the width of the surface
height	the height of the surface
format	the SDL_PixelFormatEnum for the new surface's pixel format.

Return Value

Returns the new SDL_Surface structure that is created or NULL if it fails; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateSurfaceFrom
- SDL_DestroySurface

SDL_CreateSurfaceFrom

Allocate a new RGB surface with a specific pixel format and existing pixel data.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_Surface* SDL_CreateSurfaceFrom(void *pixels, int width, int height,
    int pitch, SDL_PixelFormatEnum format);
```

Function Parameters

pixels	a pointer to existing pixel data
width	the width of the surface
height	the height of the surface
pitch	the number of bytes between each row, including padding
format	the SDL_PixelFormatEnum for the new surface's pixel format.

Return Value

Returns the new SDL_Surface structure that is created or NULL if it fails; call SDL_GetError() for more information.

Remarks

No copy is made of the pixel data. Pixel data is not managed automatically; you must free the surface before you free the pixel data.

Pitch is the offset in bytes from one row of pixels to the next, e.g. `width*4` for `SDL_PIXELFORMAT_RGBA8888`.

You may pass NULL for pixels and 0 for pitch to create a surface that you will fill in with valid values later.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateSurface
- SDL_DestroySurface

SDL_CreateSystemCursor

Create a system cursor.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_Cursor* SDL_CreateSystemCursor(SDL_SystemCursor id);
```

Function Parameters

id	an SDL_SystemCursor enum value

Return Value

Returns a cursor on success or NULL on failure; call SDL_GetError() for more information.

Code Examples

```
SDL_Cursor* cursor;
cursor = SDL_CreateSystemCursor(SDL_SYSTEM_CURSOR_HAND);
SDL_SetCursor(cursor);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroyCursor

SDL_CreateTexture

Create a texture for a rendering context.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Texture* SDL_CreateTexture(SDL_Renderer *renderer,
    SDL_PixelFormatEnum format, int access, int w, int h);
```

Function Parameters

renderer	the rendering context
format	one of the enumerated values in SDL_PixelFormatEnum
access	one of the enumerated values in SDL_TextureAccess
w	the width of the texture in pixels
h	the height of the texture in pixels

Return Value

Returns a pointer to the created texture or NULL if no rendering context was active, the format was unsupported, or the width or height were out of range; call SDL_GetError() for more information.

Code Examples

```
#include <SDL3/SDL.h>

#include <stdlib.h>

/* Moving Rectangle */
int main(int argc, char *argv[])
{
        SDL_Window *window;
        SDL_Renderer *renderer;
        SDL_Texture *texture;
        SDL_Event event;
        SDL_FRect r;

        if (SDL_Init(SDL_INIT_VIDEO) < 0) {
                SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't
                    initialize SDL: %s", SDL_GetError());
                return 3;
```

```
        }

        window = SDL_CreateWindow("SDL_CreateTexture",
                       1024, 768,
                       SDL_WINDOW_RESIZABLE);

        r.w = 100;
        r.h = 50;

        renderer = SDL_CreateRenderer(window, NULL, 0);

        texture = SDL_CreateTexture(renderer, SDL_PIXELFORMAT_RGBA8888,
            SDL_TEXTUREACCESS_TARGET, 1024, 768);

        while (1) {
                SDL_PollEvent(&event);
                if(event.type == SDL_EVENT_QUIT)
                        break;
                r.x=rand()%500;
                r.y=rand()%500;

                SDL_SetRenderTarget(renderer, texture);
                SDL_SetRenderDrawColor(renderer, 0x00, 0x00, 0x00, 0x00);
                SDL_RenderClear(renderer);
                SDL_RenderRect(renderer,&r);
                SDL_SetRenderDrawColor(renderer, 0xFF, 0x00, 0x00, 0x00);
                SDL_RenderFillRect(renderer, &r);
                SDL_SetRenderTarget(renderer, NULL);
                SDL_RenderTexture(renderer, texture, NULL, NULL);
                SDL_RenderPresent(renderer);
        }
        SDL_DestroyRenderer(renderer);
        SDL_Quit();
        return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateTextureFromSurface
- SDL_CreateTextureWithProperties
- SDL_DestroyTexture
- SDL_QueryTexture

- SDL_UpdateTexture

SDL_CreateTextureFromSurface

Create a texture from an existing surface.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Texture* SDL_CreateTextureFromSurface(SDL_Renderer *renderer,
    SDL_Surface *surface);
```

Function Parameters

renderer	the rendering context
surface	the SDL_Surface structure containing pixel data used to fill the texture

Return Value

Returns the created texture or NULL on failure; call SDL_GetError() for more information.

Remarks

The surface is not modified or freed by this function.

The SDL_TextureAccess hint for the created texture is `SDL_TEXTUREACCESS_STATIC`.

The pixel format of the created texture may be different from the pixel format of the surface. Use SDL_QueryTexture() to query the pixel format of the texture.

Code Examples

```
SDL_Renderer *renderer;

SDL_Surface *surface = SDL_CreateSurface(640, 480,
    SDL_PIXELFORMAT_RGBA8888);

if (surface == NULL) {
    fprintf(stderr, "CreateRGBSurface failed: %s\n", SDL_GetError());
    exit(1);
}

SDL_Texture *texture = SDL_CreateTextureFromSurface(renderer, surface);
```

```
if (texture == NULL) {
    fprintf(stderr, "CreateTextureFromSurface failed: %s\n",
        SDL_GetError());
    exit(1);
}

SDL_DestroySurface(surface);
surface = NULL;
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateTexture
- SDL_CreateTextureWithProperties
- SDL_DestroyTexture
- SDL_QueryTexture

SDL_CreateTextureWithProperties

Create a texture for a rendering context with the specified properties.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Texture* SDL_CreateTextureWithProperties(SDL_Renderer *renderer,
    SDL_PropertiesID props);
```

Function Parameters

renderer	the rendering context
props	the properties to use

Return Value

Returns a pointer to the created texture or NULL if no rendering context was active, the format was unsupported, or the width or height were out of range; call SDL_GetError() for more information.

Remarks

These are the supported properties:

- `SDL_PROP_TEXTURE_CREATE_COLORSPACE_NUMBER`: an SDL_ColorSpace value describing the texture colorspace, defaults to SDL_COLORSPACE_SRGB_LINEAR for floating point textures, SDL_COLORSPACE_HDR10 for 10-bit textures, SDL_COLORSPACE_SRGB for other RGB textures and SDL_COLORSPACE_JPEG for YUV textures.
- `SDL_PROP_TEXTURE_CREATE_FORMAT_NUMBER`: one of the enumerated values in SDL_PixelFormatEnum, defaults to the best RGBA format for the renderer
- `SDL_PROP_TEXTURE_CREATE_ACCESS_NUMBER`: one of the enumerated values in SDL_TextureAccess, defaults to SDL_TEXTUREACCESS_STATIC
- `SDL_PROP_TEXTURE_CREATE_WIDTH_NUMBER`: the width of the texture in pixels, required
- `SDL_PROP_TEXTURE_CREATE_HEIGHT_NUMBER`: the height of the texture in pixels, required

- `SDL_PROP_TEXTURE_CREATE_SDR_WHITE_POINT_FLOAT`: for HDR10 and floating point textures, this defines the value of 100diffuse white, with higher values being displayed in the High Dynamic Range headroom. This defaults to 100 for HDR10 textures and 1.0 for floating point textures.
- `SDL_PROP_TEXTURE_CREATE_HDR_HEADROOM_FLOAT`: for HDR10 and floating point textures, this defines the maximum dynamic range used by the content, in terms of the SDR white point. This would be equivalent to maxCLL / SDL_PROP_TEXTURE_CREATE_SDR_WHITE_POINT_FLOAT for HDR10 content. If this is defined, any values outside the range supported by the display will be scaled into the available HDR headroom, otherwise they are clipped.

With the direct3d11 renderer:

- `SDL_PROP_TEXTURE_CREATE_D3D11_TEXTURE_POINTER`: the ID3D11Texture2D associated with the texture, if you want to wrap an existing texture.
- `SDL_PROP_TEXTURE_CREATE_D3D11_TEXTURE_U_POINTER`: the ID3D11Texture2D associated with the U plane of a YUV texture, if you want to wrap an existing texture.
- `SDL_PROP_TEXTURE_CREATE_D3D11_TEXTURE_V_POINTER`: the ID3D11Texture2D associated with the V plane of a YUV texture, if you want to wrap an existing texture.

With the direct3d12 renderer:

- `SDL_PROP_TEXTURE_CREATE_D3D12_TEXTURE_POINTER`: the ID3D12Resource associated with the texture, if you want to wrap an existing texture.
- `SDL_PROP_TEXTURE_CREATE_D3D12_TEXTURE_U_POINTER`: the ID3D12Resource associated with the U plane of a YUV texture, if you want to wrap an existing texture.
- `SDL_PROP_TEXTURE_CREATE_D3D12_TEXTURE_V_POINTER`: the ID3D12Resource associated with the V plane of a YUV texture, if you want to wrap an existing texture.

With the metal renderer:

- `SDL_PROP_TEXTURE_CREATE_METAL_PIXELBUFFER_POINTER`: the CVPixelBufferRef associated with the texture, if you want to create a texture from an existing pixel buffer.

With the opengl renderer:

- `SDL_PROP_TEXTURE_CREATE_OPENGL_TEXTURE_NUMBER`: the GLuint texture associated with the texture, if you want to wrap an existing texture.

- SDL_PROP_TEXTURE_CREATE_OPENGL_TEXTURE_UV_NUMBER: the GLuint texture associated with the UV plane of an NV12 texture, if you want to wrap an existing texture.
- SDL_PROP_TEXTURE_CREATE_OPENGL_TEXTURE_U_NUMBER: the GLuint texture associated with the U plane of a YUV texture, if you want to wrap an existing texture.
- SDL_PROP_TEXTURE_CREATE_OPENGL_TEXTURE_V_NUMBER: the GLuint texture associated with the V plane of a YUV texture, if you want to wrap an existing texture.

With the opengles2 renderer:

- SDL_PROP_TEXTURE_CREATE_OPENGLES2_TEXTURE_NUMBER: the GLuint texture associated with the texture, if you want to wrap an existing texture.
- SDL_PROP_TEXTURE_CREATE_OPENGLES2_TEXTURE_NUMBER: the GLuint texture associated with the texture, if you want to wrap an existing texture.
- SDL_PROP_TEXTURE_CREATE_OPENGLES2_TEXTURE_UV_NUMBER: the GLuint texture associated with the UV plane of an NV12 texture, if you want to wrap an existing texture.
- SDL_PROP_TEXTURE_CREATE_OPENGLES2_TEXTURE_U_NUMBER: the GLuint texture associated with the U plane of a YUV texture, if you want to wrap an existing texture.
- SDL_PROP_TEXTURE_CREATE_OPENGLES2_TEXTURE_V_NUMBER: the GLuint texture associated with the V plane of a YUV texture, if you want to wrap an existing texture.

With the vulkan renderer:

- SDL_PROP_TEXTURE_CREATE_VULKAN_TEXTURE_NUMBER: the VkImage with layout VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL associated with the texture, if you want to wrap an existing texture.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateProperties
- SDL_CreateTexture
- SDL_CreateTextureFromSurface

- SDL_DestroyTexture
- SDL_QueryTexture
- SDL_UpdateTexture

SDL_CreateThread

Create a new thread with a default stack size.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
SDL_Thread * SDL_CreateThread(SDL_ThreadFunction fn, const char *name,
    void *data);
```

Function Parameters

fn	the SDL_ThreadFunction function to call in the new thread
name	the name of the thread
data	a pointer that is passed to `fn`

Return Value

Returns an opaque pointer to the new thread object on success, NULL if the new thread could not be created; call SDL_GetError() for more information.

Remarks

This is equivalent to calling:

```
SDL_CreateThreadWithStackSize(fn, name, 0, data);
```

Code Examples

```
#include <SDL3/SDL.h>

/* Very simple thread - counts 0 to 9 delaying 50ms between increments */
static int TestThread(void *ptr)
{
    int cnt;

    for (cnt = 0; cnt < 10; ++cnt) {
        SDL_Log("Thread counter: %d\n", cnt);
        SDL_Delay(50);
    }

    return cnt;
```

```
}

int main(int argc, char *argv[])
{
    SDL_Thread *thread;
    int        threadReturnValue;

    SDL_Log("Simple SDL_CreateThread test:\n");

    /* Simply create a thread */
    thread = SDL_CreateThread(TestThread, "TestThread", (void *)NULL);

    if (NULL == thread) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "SDL_CreateThread
            failed: %s\n", SDL_GetError());
    } else {
        SDL_WaitThread(thread, &threadReturnValue);
        SDL_Log("Thread returned value: %d\n", threadReturnValue);
    }

    return 0;
}
```

```
Output:
Simple SDL_CreateThread test:
Thread counter: 0
Thread counter: 1
Thread counter: 2
Thread counter: 3
Thread counter: 4
Thread counter: 5
Thread counter: 6
Thread counter: 7
Thread counter: 8
Thread counter: 9
Thread returned value: 10
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateThreadWithStackSize
- SDL_WaitThread

SDL_CreateThreadWithStackSize

Create a new thread with a specific stack size.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
SDL_Thread * SDL_CreateThreadWithStackSize(SDL_ThreadFunction fn, const
    char *name, const size_t stacksize, void *data);
```

Function Parameters

fn	the SDL_ThreadFunction function to call in the new thread
name	the name of the thread
stacksize	the size, in bytes, to allocate for the new thread stack.
data	a pointer that is passed to `fn`

Return Value

Returns an opaque pointer to the new thread object on success, NULL if the new thread could not be created; call SDL_GetError() for more information.

Remarks

SDL makes an attempt to report `name` to the system, so that debuggers can display it. Not all platforms support this.

Thread naming is a little complicated: Most systems have very small limits for the string length (Haiku has 32 bytes, Linux currently has 16, Visual C++ 6.0 has _nine_!), and possibly other arbitrary rules. You'll have to see what happens with your system's debugger. The name should be UTF-8 (but using the naming limits of C identifiers is a better bet). There are no requirements for thread naming conventions, so long as the string is null-terminated UTF-8, but these guidelines are helpful in choosing a name:

https://stackoverflow.com/questions/149932/naming-conventions-for-threads

If a system imposes requirements, SDL will try to munge the string for it (truncate, etc), but the original string contents will be available from SDL_GetThreadName().

The size (in bytes) of the new stack can be specified. Zero means "use the system default" which might be wildly different between platforms. x86 Linux generally defaults to eight megabytes, an embedded device might be a few kilobytes instead. You generally need to specify a stack that is a multiple of the system's page size (in many cases, this is 4 kilobytes, but check your system documentation).

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateThread
- SDL_WaitThread

SDL_CreateTLS

Create a piece of thread-local storage.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
SDL_TLSID SDL_CreateTLS(void);
```

Return Value

Returns the newly created thread local storage identifier or 0 on error.

Remarks

This creates an identifier that is globally visible to all threads but refers to data that is thread-specific.

Code Examples

```
// BEWARE: This code example was migrated from the SDL2 Wiki, by only
    updating the names.

static SDL_SpinLock tls_lock;
static SDL_TLSID thread_local_storage;
void SetMyThreadData(void *value)
{
    if (!thread_local_storage) {
        SDL_LockSpinlock(&tls_lock);
        if (!thread_local_storage) {
            thread_local_storage = SDL_CreateTLS();
        }
        SDL_UnlockSpinlock(&tls_lock);
    }
    SDL_SetTLS(thread_local_storage, value, 0);
}
void *GetMyThreadData(void)
{
    return SDL_GetTLS(thread_local_storage);
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTLS
- SDL_SetTLS

SDL_CreateWindow

Create a window with the specified dimensions and flags.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_Window* SDL_CreateWindow(const char *title, int w, int h,
    SDL_WindowFlags flags);
```

Function Parameters

title	the title of the window, in UTF-8 encoding
w	the width of the window
h	the height of the window
flags	0, or one or more SDL_WindowFlags OR'd together

Return Value

Returns the window that was created or NULL on failure; call SDL_GetError() for more information.

Remarks

`flags` may be any of the following OR'd together:

- `SDL_WINDOW_FULLSCREEN`: fullscreen window at desktop resolution
- `SDL_WINDOW_OPENGL`: window usable with an OpenGL context
- `SDL_WINDOW_VULKAN`: window usable with a Vulkan instance
- `SDL_WINDOW_METAL`: window usable with a Metal instance
- `SDL_WINDOW_HIDDEN`: window is not visible
- `SDL_WINDOW_BORDERLESS`: no window decoration
- `SDL_WINDOW_RESIZABLE`: window can be resized
- `SDL_WINDOW_MINIMIZED`: window is minimized
- `SDL_WINDOW_MAXIMIZED`: window is maximized
- `SDL_WINDOW_MOUSE_GRABBED`: window has grabbed mouse focus

The SDL_Window is implicitly shown if SDL_WINDOW_HIDDEN is not set.

On Apple's macOS, you **must** set the NSHighResolutionCapable Info.plist property to YES, otherwise you will not receive a High-DPI OpenGL canvas.

The window pixel size may differ from its window coordinate size if the window is on a high pixel density display. Use SDL_GetWindowSize() to query the client area's size in window coordinates, and SDL_GetWindowSizeInPixels() or SDL_GetRenderOutputSize() to query the drawable size in pixels. Note that the drawable size can vary after the window is created and should be queried again if you get an SDL_EVENT_WINDOW_PIXEL_SIZE_CHANGED event.

If the window is created with any of the SDL_WINDOW_OPENGL or SDL_WINDOW_VULKAN flags, then the corresponding LoadLibrary function (SDL_GL_LoadLibrary or SDL_Vulkan_LoadLibrary) is called and the corresponding UnloadLibrary function is called by SDL_DestroyWindow().

If SDL_WINDOW_VULKAN is specified and there isn't a working Vulkan driver, SDL_CreateWindow() will fail because SDL_Vulkan_LoadLibrary() will fail.

If SDL_WINDOW_METAL is specified on an OS that does not support Metal, SDL_CreateWindow() will fail.

On non-Apple devices, SDL requires you to either not link to the Vulkan loader or link to a dynamic library version. This limitation may be removed in a future version of SDL.

Code Examples

```
// Example program:
// Using SDL3 to create an application window

#include <SDL3/SDL.h>

int main(int argc, char* argv[]) {

    SDL_Window *window;                // Declare a pointer

    SDL_Init(SDL_INIT_VIDEO);          // Initialize SDL2

    // Create an application window with the following settings:
    window = SDL_CreateWindow(
        "An SDL3 window",              // window title
        640,                           // width, in pixels
        480,                           // height, in pixels
        SDL_WINDOW_OPENGL              // flags - see below
    );

    // Check that the window was successfully created
    if (window == NULL) {
        // In the case that the window could not be made...
```

```
        SDL_LogError(SDL_LOG_CATEGORY_ERROR, "Could not create window:
            %s\n", SDL_GetError());
        return 1;
    }

    // The window is open: could enter program loop here (see
        SDL_PollEvent())

    SDL_Delay(3000); // Pause execution for 3000 milliseconds, for
        example

    // Close and destroy the window
    SDL_DestroyWindow(window);

    // Clean up
    SDL_Quit();
    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreatePopupWindow
- SDL_CreateWindowWithProperties
- SDL_DestroyWindow

SDL_CreateWindowAndRenderer

Create a window and default renderer.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_CreateWindowAndRenderer(const char *title, int width, int
    height, SDL_WindowFlags window_flags, SDL_Window **window,
    SDL_Renderer **renderer);
```

Function Parameters

title	the title of the window, in UTF-8 encoding
width	the width of the window
height	the height of the window
window_flags	the flags used to create the window (see SDL_CreateWindow())
window	a pointer filled with the window, or NULL on error
renderer	a pointer filled with the renderer, or NULL on error

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Code Examples

```
#include <SDL3/SDL.h>

int main(int argc, char *argv[])
{
    SDL_Window *window;
    SDL_Renderer *renderer;
    SDL_Surface *surface;
    SDL_Texture *texture;
    SDL_Event event;

    if (SDL_Init(SDL_INIT_VIDEO) < 0) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't initialize
            SDL: %s", SDL_GetError());
```

```
        return 3;
    }

    if (SDL_CreateWindowAndRenderer("Hello SDL", 320, 240,
         SDL_WINDOW_RESIZABLE, &window, &renderer)) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create
            window and renderer: %s", SDL_GetError());
        return 3;
    }

    surface = SDL_LoadBMP("sample.bmp");
    if (!surface) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create
            surface from image: %s", SDL_GetError());
        return 3;
    }
    texture = SDL_CreateTextureFromSurface(renderer, surface);
    if (!texture) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create
            texture from surface: %s", SDL_GetError());
        return 3;
    }
    SDL_DestroySurface(surface);

    while (1) {
        SDL_PollEvent(&event);
        if (event.type == SDL_EVENT_QUIT) {
            break;
        }
        SDL_SetRenderDrawColor(renderer, 0x00, 0x00, 0x00, 0x00);
        SDL_RenderClear(renderer);
        SDL_RenderTexture(renderer, texture, NULL, NULL);
        SDL_RenderPresent(renderer);
    }

    SDL_DestroyTexture(texture);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);

    SDL_Quit();

    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateRenderer
- SDL_CreateWindow

SDL_CreateWindowWithProperties

Create a window with the specified properties.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_Window* SDL_CreateWindowWithProperties(SDL_PropertiesID props);
```

Function Parameters

props	the properties to use

Return Value

Returns the window that was created or NULL on failure; call SDL_GetError() for more information.

Remarks

These are the supported properties:

- `SDL_PROP_WINDOW_CREATE_ALWAYS_ON_TOP_BOOLEAN`: true if the window should be always on top
- `SDL_PROP_WINDOW_CREATE_BORDERLESS_BOOLEAN`: true if the window has no window decoration
- `SDL_PROP_WINDOW_CREATE_EXTERNAL_GRAPHICS_CONTEXT_BOOLEAN`: true if the window will be used with an externally managed graphics context.
- `SDL_PROP_WINDOW_CREATE_FOCUSABLE_BOOLEAN`: true if the window should accept keyboard input (defaults true)
- `SDL_PROP_WINDOW_CREATE_FULLSCREEN_BOOLEAN`: true if the window should start in fullscreen mode at desktop resolution
- `SDL_PROP_WINDOW_CREATE_HEIGHT_NUMBER`: the height of the window
- `SDL_PROP_WINDOW_CREATE_HIDDEN_BOOLEAN`: true if the window should start hidden
- `SDL_PROP_WINDOW_CREATE_HIGH_PIXEL_DENSITY_BOOLEAN`: true if the window uses a high pixel density buffer if possible

- SDL_PROP_WINDOW_CREATE_MAXIMIZED_BOOLEAN: true if the window should start maximized
- SDL_PROP_WINDOW_CREATE_MENU_BOOLEAN: true if the window is a popup menu
- SDL_PROP_WINDOW_CREATE_METAL_BOOLEAN: true if the window will be used with Metal rendering
- SDL_PROP_WINDOW_CREATE_MINIMIZED_BOOLEAN: true if the window should start minimized
- SDL_PROP_WINDOW_CREATE_MODAL_BOOLEAN: true if the window is modal to its parent
- SDL_PROP_WINDOW_CREATE_MOUSE_GRABBED_BOOLEAN: true if the window starts with grabbed mouse focus
- SDL_PROP_WINDOW_CREATE_OPENGL_BOOLEAN: true if the window will be used with OpenGL rendering
- SDL_PROP_WINDOW_CREATE_PARENT_POINTER: an SDL_Window that will be the parent of this window, required for windows with the "toolip", "menu", and "modal" properties
- SDL_PROP_WINDOW_CREATE_RESIZABLE_BOOLEAN: true if the window should be resizable
- SDL_PROP_WINDOW_CREATE_TITLE_STRING: the title of the window, in UTF-8 encoding
- SDL_PROP_WINDOW_CREATE_TRANSPARENT_BOOLEAN: true if the window show transparent in the areas with alpha of 0
- SDL_PROP_WINDOW_CREATE_TOOLTIP_BOOLEAN: true if the window is a tooltip
- SDL_PROP_WINDOW_CREATE_UTILITY_BOOLEAN: true if the window is a utility window, not showing in the task bar and window list
- SDL_PROP_WINDOW_CREATE_VULKAN_BOOLEAN: true if the window will be used with Vulkan rendering
- SDL_PROP_WINDOW_CREATE_WIDTH_NUMBER: the width of the window
- SDL_PROP_WINDOW_CREATE_X_NUMBER: the x position of the window, or SDL_WINDOWPOS_CENTERED, defaults to SDL_WINDOWPOS_UNDEFINED. This is relative to the parent for windows with the "parent" property set.
- SDL_PROP_WINDOW_CREATE_Y_NUMBER: the y position of the window, or SDL_WINDOWPOS_CENTERED, defaults to SDL_WINDOWPOS_UNDEFINED. This is relative to the parent for windows with the "parent" property set.

These are additional supported properties on macOS:

- `SDL_PROP_WINDOW_CREATE_COCOA_WINDOW_POINTER`: the `(__unsafe_unretained)` NSWindow associated with the window, if you want to wrap an existing window.
- `SDL_PROP_WINDOW_CREATE_COCOA_VIEW_POINTER`: the `(__unsafe_unretained)` NSView associated with the window, defaults to `[window contentView]`

These are additional supported properties on Wayland:

- `SDL_PROP_WINDOW_CREATE_WAYLAND_SCALE_TO_DISPLAY_BOOLEAN` - true if the window should use forced scaling designed to produce 1:1 pixel mapping if not flagged as being DPI-aware. This is intended to allow legacy applications to be displayed without desktop scaling being applied, and has issues with certain display configurations, as this forces the window to behave in a way that Wayland desktops were not designed to accommodate. Potential issues include, but are not limited to: rounding errors can result when odd window sizes/scales are used, the window may be unusably small, the window may jump in visible size at times, the window may appear to be larger than the desktop space, and possible loss of cursor precision can occur. New applications should be designed with proper DPI awareness and handling instead of enabling this.
- `SDL_PROP_WINDOW_CREATE_WAYLAND_SURFACE_ROLE_CUSTOM_BOOLEAN` - true if the application wants to use the Wayland surface for a custom role and does not want it attached to an XDG toplevel window. See README/wayland for more information on using custom surfaces.
- `SDL_PROP_WINDOW_CREATE_WAYLAND_CREATE_EGL_WINDOW_BOOLEAN` - true if the application wants an associated `wl_egl_window` object to be created, even if the window does not have the OpenGL property or flag set.
- `SDL_PROP_WINDOW_CREATE_WAYLAND_WL_SURFACE_POINTER` - the wl_surface associated with the window, if you want to wrap an existing window. See README/wayland for more information.

These are additional supported properties on Windows:

- `SDL_PROP_WINDOW_CREATE_WIN32_HWND_POINTER`: the HWND associated with the window, if you want to wrap an existing window.
- `SDL_PROP_WINDOW_CREATE_WIN32_PIXEL_FORMAT_HWND_POINTER`: optional, another window to share pixel format with, useful for OpenGL windows

These are additional supported properties with X11:

- `SDL_PROP_WINDOW_CREATE_X11_WINDOW_NUMBER`: the X11 Window associated with the window, if you want to wrap an existing window.

The window is implicitly shown if the "hidden" property is not set.

Windows with the "tooltip" and "menu" properties are popup windows and have the behaviors and guidelines outlined in SDL_CreatePopupWindow().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateProperties
- SDL_CreateWindow
- SDL_DestroyWindow

SDL_CursorVisible

Return whether the cursor is currently being shown.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_bool SDL_CursorVisible(void);
```

Return Value

Returns `SDL_TRUE` if the cursor is being shown, or `SDL_FALSE` if the cursor is hidden.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HideCursor
- SDL_ShowCursor

SDL_DateTimeToTime

Converts a calendar time to an SDL_Time in nanoseconds since the epoch.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
int SDL_DateTimeToTime(const SDL_DateTime *dt, SDL_Time *ticks);
```

Function Parameters

dt	the source SDL_DateTime
ticks	the resulting SDL_Time

Return Value

Returns 0 on success or -1 on error; call SDL_GetError() for more information.

Remarks

This function ignores the day_of_week member of the SDL_DateTime struct, so it may remain unset.

Version

This function is available since SDL 3.0.0.

SDL_Delay

Wait a specified number of milliseconds before returning.

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
void SDL_Delay(Uint32 ms);
```

Function Parameters

ms	the number of milliseconds to delay

Remarks

This function waits a specified number of milliseconds before returning. It waits at least the specified time, but possibly longer due to OS scheduling.

Version

This function is available since SDL 3.0.0.

SDL_DelayNS

Wait a specified number of nanoseconds before returning.

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
void SDL_DelayNS(Uint64 ns);
```

Function Parameters

ns	the number of nanoseconds to delay

Remarks

This function waits a specified number of nanoseconds before returning. It waits at least the specified time, but possibly longer due to OS scheduling.

Version

This function is available since SDL 3.0.0.

SDL_DelEventWatch

Remove an event watch callback added with SDL_AddEventWatch().

Header File

Defined in SDL3/SDL_events.h

Syntax

```
void SDL_DelEventWatch(SDL_EventFilter filter, void *userdata);
```

Function Parameters

filter	the function originally passed to SDL_AddEventWatch()
userdata	the pointer originally passed to SDL_AddEventWatch()

Remarks

This function takes the same input as SDL_AddEventWatch() to identify and delete the corresponding callback.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AddEventWatch

SDL_DelHintCallback

Remove a function watching a particular hint.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
void SDL_DelHintCallback(const char *name,
                         SDL_HintCallback callback,
                         void *userdata);
```

Function Parameters

name	the hint being watched
callback	An SDL_HintCallback function that will be called when the hint value changes
userdata	a pointer being passed to the callback function

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AddHintCallback

SDL_DestroyAudioStream

Free an audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
void SDL_DestroyAudioStream(SDL_AudioStream *stream);
```

Function Parameters

stream	The audio stream to destroy.

Remarks

This will release all allocated data, including any audio that is still queued. You do not need to manually clear the stream first.

If this stream was bound to an audio device, it is unbound during this call. If this stream was created with SDL_OpenAudioDeviceStream, the audio device that was opened alongside this stream's creation will be closed, too.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateAudioStream

SDL_DestroyCondition

Destroy a condition variable.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
void SDL_DestroyCondition(SDL_Condition *cond);
```

Function Parameters

cond	the condition variable to destroy

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateCondition

SDL_DestroyCursor

Free a previously-created cursor.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
void SDL_DestroyCursor(SDL_Cursor * cursor);
```

Function Parameters

cursor	the cursor to free

Remarks

Use this function to free cursor resources created with SDL_CreateCursor(), SDL_CreateColorCursor() or SDL_CreateSystemCursor().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateColorCursor
- SDL_CreateCursor
- SDL_CreateSystemCursor

SDL_DestroyHapticEffect

Destroy a haptic effect on the device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
void SDL_DestroyHapticEffect(SDL_Haptic *haptic, int effect);
```

Function Parameters

haptic	the SDL_Haptic device to destroy the effect on
effect	the ID of the haptic effect to destroy

Remarks

This will stop the effect if it's running. Effects are automatically destroyed when the device is closed.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateHapticEffect

SDL_DestroyMutex

Destroy a mutex created with SDL_CreateMutex().

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
void SDL_DestroyMutex(SDL_Mutex *mutex);
```

Function Parameters

mutex	the mutex to destroy

Remarks

This function must be called on any mutex that is no longer needed. Failure to destroy a mutex will result in a system memory or resource leak. While it is safe to destroy a mutex that is _unlocked_, it is not safe to attempt to destroy a locked mutex, and may result in undefined behavior depending on the platform.

Code Examples

¡¡Include(SDL_CreateMutex, , , from="## Begin Mutex Example", to="## End Mutex Example")¿¿

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateMutex

SDL_DestroyPalette

Free a palette created with SDL_CreatePalette().

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
void SDL_DestroyPalette(SDL_Palette * palette);
```

Function Parameters

palette	the SDL_Palette structure to be freed

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreatePalette

SDL_DestroyPixelFormat

Free an SDL_PixelFormat structure allocated by SDL_CreatePixelFormat().

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
void SDL_DestroyPixelFormat(SDL_PixelFormat *format);
```

Function Parameters

format	the SDL_PixelFormat structure to free

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreatePixelFormat

SDL_DestroyProperties

Destroy a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
void SDL_DestroyProperties(SDL_PropertiesID props);
```

Function Parameters

props	the properties to destroy

Remarks

All properties are deleted and their cleanup functions will be called, if any.

Thread Safety

This function should not be called while these properties are locked or other threads might be setting or getting values from these properties.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateProperties

SDL_DestroyRenderer

Destroy the rendering context for a window and free associated textures.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
void SDL_DestroyRenderer(SDL_Renderer *renderer);
```

Function Parameters

renderer	the rendering context

Remarks

If `renderer` is NULL, this function will return immediately after setting the SDL error message to "Invalid renderer". See SDL_GetError().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateRenderer

SDL_DestroyRWLock

Destroy a read/write lock created with SDL_CreateRWLock().

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
void SDL_DestroyRWLock(SDL_RWLock *rwlock);
```

Function Parameters

rwlock	the rwlock to destroy

Remarks

This function must be called on any read/write lock that is no longer needed. Failure to destroy a rwlock will result in a system memory or resource leak. While it is safe to destroy a rwlock that is _unlocked_, it is not safe to attempt to destroy a locked rwlock, and may result in undefined behavior depending on the platform.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateRWLock

SDL_DestroySemaphore

Destroy a semaphore.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
void SDL_DestroySemaphore(SDL_Semaphore *sem);
```

Function Parameters

sem	the semaphore to destroy

Remarks

It is not safe to destroy a semaphore if there are threads currently waiting on it.

Code Examples

¡¡Include(SDL_CreateSemaphore, , , from="## Begin Semaphore Example", to="## End Semaphore Example")¿¿

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateSemaphore

SDL_DestroySurface

Free an RGB surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
void SDL_DestroySurface(SDL_Surface *surface);
```

Function Parameters

surface	the SDL_Surface to free.

Remarks

It is safe to pass NULL to this function.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateSurface
- SDL_CreateSurfaceFrom

SDL_DestroyTexture

Destroy the specified texture.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
void SDL_DestroyTexture(SDL_Texture *texture);
```

Function Parameters

texture	the texture to destroy

Remarks

Passing NULL or an otherwise invalid texture will set the SDL error message to "Invalid texture".

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateTexture
- SDL_CreateTextureFromSurface

SDL_DestroyWindow

Destroy a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
void SDL_DestroyWindow(SDL_Window *window);
```

Function Parameters

window	the window to destroy

Remarks

Any popups or modal windows owned by the window will be recursively destroyed as well.

If `window` is NULL, this function will return immediately after setting the SDL error message to "Invalid window". See SDL_GetError().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreatePopupWindow
- SDL_CreateWindow
- SDL_CreateWindowWithProperties

SDL_DestroyWindowSurface

Destroy the surface associated with the window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_DestroyWindowSurface(SDL_Window *window);
```

Function Parameters

window	the window to update

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowSurface
- SDL_WindowHasSurface

SDL_DetachThread

Let a thread clean up on exit without intervention.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
void SDL_DetachThread(SDL_Thread * thread);
```

Function Parameters

thread	the SDL_Thread pointer that was returned from the SDL_CreateThread() call that started this thread

Remarks

A thread may be "detached" to signify that it should not remain until another thread has called SDL_WaitThread() on it. Detaching a thread is useful for long-running threads that nothing needs to synchronize with or further manage. When a detached thread is done, it simply goes away.

There is no way to recover the return code of a detached thread. If you need this, don't detach the thread and instead use SDL_WaitThread().

Once a thread is detached, you should usually assume the SDL_Thread isn't safe to reference again, as it will become invalid immediately upon the detached thread's exit, instead of remaining until someone has called SDL_WaitThread() to finally clean it up. As such, don't detach the same thread more than once.

If a thread has already exited when passed to SDL_DetachThread(), it will stop waiting for a call to SDL_WaitThread() and clean up immediately. It is not safe to detach a thread that might be used with SDL_WaitThread().

You may not call SDL_WaitThread() on a thread that has been detached. Use either that function or this one, but not both, or behavior is undefined.

It is safe to pass NULL to this function; it is a no-op.

Code Examples

```
extern int TestThread(void *ptr);
SDL_Thread *thread = SDL_CreateThread(TestThread, "TestThread", (void
    *)NULL);
SDL_DetachThread(thread); /* will go away on its own upon completion. */
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateThread
- SDL_WaitThread

SDL_DetachVirtualJoystick

Detach a virtual joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_DetachVirtualJoystick(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID, previously returned from SDL_AttachVirtualJoystick()

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AttachVirtualJoystick
- SDL_AttachVirtualJoystickEx

SDL_Direct3D9GetAdapterIndex

Get the D3D9 adapter index that matches the specified display.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_Direct3D9GetAdapterIndex(SDL_DisplayID displayID);
```

Function Parameters

displayID	the instance of the display to query

Return Value

Returns the D3D9 adapter index on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The returned adapter index can be passed to `IDirect3D9::CreateDevice` and controls on which monitor a full screen application will appear.

Version

This function is available since SDL 3.0.0.

SDL_DisableScreenSaver

Prevent the screen from being blanked by a screen saver.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_DisableScreenSaver(void);
```

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If you disable the screensaver, it is automatically re-enabled when SDL quits.

The screensaver is disabled by default, but this may by changed by SDL_HINT_VIDEO_ALLOW_SCRI

Version

This function is available since SDL 3.0.0.

See Also

- SDL_EnableScreenSaver
- SDL_ScreenSaverEnabled

SDL_DuplicateSurface

Creates a new surface identical to the existing surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_Surface* SDL_DuplicateSurface(SDL_Surface *surface);
```

Function Parameters

surface	the surface to duplicate.

Return Value

Returns a copy of the surface, or NULL on failure; call SDL_GetError() for more information.

Remarks

The returned surface should be freed with SDL_DestroySurface().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroySurface

SDL_DXGIGetOutputInfo

Get the DXGI Adapter and Output indices for the specified display.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
SDL_bool SDL_DXGIGetOutputInfo(SDL_DisplayID displayID, int
    *adapterIndex, int *outputIndex);
```

Function Parameters

displayID	the instance of the display to query
adapterIndex	a pointer to be filled in with the adapter index
outputIndex	a pointer to be filled in with the output index

Return Value

Returns SDL_TRUE on success or SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

The DXGI Adapter and Output indices can be passed to `EnumAdapters` and `EnumOutputs` respectively to get the objects required to create a DX10 or DX11 device and swap chain.

Version

This function is available since SDL 3.0.0.

SDL_EGL_GetCurrentEGLConfig

Get the currently active EGL config.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_EGLConfig SDL_EGL_GetCurrentEGLConfig(void);
```

Return Value

Returns the currently active EGL config or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_EGL_GetCurrentEGLDisplay

Get the currently active EGL display.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_EGLDisplay SDL_EGL_GetCurrentEGLDisplay(void);
```

Return Value

Returns the currently active EGL display or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_EGL_GetProcAddress

Get an EGL library function by name.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_FunctionPointer SDL_EGL_GetProcAddress(const char *proc);
```

Function Parameters

proc	the name of an EGL function

Return Value

Returns a pointer to the named EGL function. The returned pointer should be cast to the appropriate function signature.

Remarks

If an EGL library is loaded, this function allows applications to get entry points for EGL functions. This is useful to provide to an EGL API and extension loader.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_GetCurrentEGLDisplay

SDL_EGL_GetWindowEGLSurface

Get the EGL surface associated with the window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_EGLSurface SDL_EGL_GetWindowEGLSurface(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the EGLSurface pointer associated with the window, or NULL on failure.

Version

This function is available since SDL 3.0.0.

SDL_EGL_SetEGLAttributeCallbacks

Sets the callbacks for defining custom EGLAttrib arrays for EGL initialization.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
void SDL_EGL_SetEGLAttributeCallbacks(SDL_EGLAttribArrayCallback
    platformAttribCallback,
                                  SDL_EGLIntArrayCallback
                                      surfaceAttribCallback,
                                  SDL_EGLIntArrayCallback
                                      contextAttribCallback);
```

Function Parameters

platformAttribCallback	Callback for attributes to pass to eglGetPlatformDisplay.
surfaceAttribCallback	Callback for attributes to pass to eglCreateSurface.
contextAttribCallback	Callback for attributes to pass to eglCreateContext.

Remarks

Each callback should return a pointer to an EGL attribute array terminated with EGL_NONE. Callbacks may return NULL pointers to signal an error, which will cause the SDL_CreateWindow process to fail gracefully.

The arrays returned by each callback will be appended to the existing attribute arrays defined by SDL.

NOTE: These callback pointers will be reset after SDL_GL_ResetAttributes.

Version

This function is available since SDL 3.0.0.

SDL_EnableScreenSaver

Allow the screen to be blanked by a screen saver.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_EnableScreenSaver(void);
```

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DisableScreenSaver
- SDL_ScreenSaverEnabled

SDL_EnterAppMainCallbacks

An entry point for SDL's use in SDL_MAIN_USE_CALLBACKS.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
int SDL_EnterAppMainCallbacks(int argc, char* argv[], SDL_AppInit_func
    appinit, SDL_AppIterate_func appiter, SDL_AppEvent_func appevent,
    SDL_AppQuit_func appquit);
```

Function Parameters

argc	standard Unix main argc
argv	standard Unix main argv
appinit	The application's SDL_AppInit function
appiter	The application's SDL_AppIterate function
appevent	The application's SDL_AppEvent function
appquit	The application's SDL_AppQuit function

Return Value

Returns standard Unix main return value

Remarks

Generally, you should not call this function directly. This only exists to hand off work into SDL as soon as possible, where it has a lot more control and functionality available, and make the inline code in SDL_main.h as small as possible.

Not all platforms use this, it's actual use is hidden in a magic header-only library, and you should not call this directly unless you _really_ know what you're doing.

Thread Safety

It is not safe to call this anywhere except as the only function call in SDL_main.

Version

This function is available since SDL 3.0.0.

SDL_EnumerateDirectory

Enumerate a directory through a callback function.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
int SDL_EnumerateDirectory(const char *path,
    SDL_EnumerateDirectoryCallback callback, void *userdata);
```

Function Parameters

path	the path of the directory to enumerate
callback	a function that is called for each entry in the directory
userdata	a pointer that is passed to `callback`

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function provides every directory entry through an app-provided callback, called once for each directory entry, until all results have been provided or the callback returns ¡= 0.

Version

This function is available since SDL 3.0.0.

SDL_EnumerateProperties

Enumerate the properties on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_EnumerateProperties(SDL_PropertiesID props,
    SDL_EnumeratePropertiesCallback callback, void *userdata);
```

Function Parameters

props	the properties to query
callback	the function to call for each property
userdata	a pointer that is passed to `callback`

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The callback function is called for each property on the set of properties. The properties are locked during enumeration.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_EnumerateStorageDirectory

Enumerate a directory in a storage container through a callback function.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
int SDL_EnumerateStorageDirectory(SDL_Storage *storage, const char
    *path, SDL_EnumerateDirectoryCallback callback, void *userdata);
```

Function Parameters

storage	a storage container
path	the path of the directory to enumerate
callback	a function that is called for each entry in the directory
userdata	a pointer that is passed to `callback`

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function provides every directory entry through an app-provided callback, called once for each directory entry, until all results have been provided or the callback returns ¡= 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StorageReady

SDL_Error

Set an SDL error from a list of error codes.

Header File

Defined in SDL3/SDL_error.h

Syntax

```
int SDL_Error(SDL_errorcode code);
```

Function Parameters

code	Error code

Return Value

Returns unconditionally -1.

Version

This function is available since SDL 3.0.0.

SDL_EventEnabled

Query the state of processing events by type.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
SDL_bool SDL_EventEnabled(Uint32 type);
```

Function Parameters

type	the type of event; see SDL_EventType for details

Return Value

Returns SDL_TRUE if the event is being processed, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetEventEnabled

SDL_exp

Compute the exponential of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_exp(double x);
```

Function Parameters

x	floating point value

Return Value

Returns value of `e^x`

Remarks

The definition of `y = exp(x)` is `y = e^x`, where `e` is the base of the natural logarithm. The inverse is the natural logarithm, SDL_log.

Domain: `-INF <= x <= INF`

Range: `0 <= y <= INF`

The output will overflow if `exp(x)` is too large to be represented.

This function operates on double-precision floating point values, use SDL_expf for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_expf
- SDL_log

SDL_expf

Compute the exponential of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_expf(float x);
```

Function Parameters

x	floating point value

Return Value

Returns value of `e^x`

Remarks

The definition of `y = exp(x)` is `y = e^x`, where `e` is the base of the natural logarithm. The inverse is the natural logarithm, SDL_logf.

Domain: `-INF <= x <= INF`

Range: `0 <= y <= INF`

The output will overflow if `exp(x)` is too large to be represented.

This function operates on single-precision floating point values, use SDL_exp for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_exp
- SDL_logf

SDL_fabs

Compute the absolute value of `x`

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_fabs(double x);
```

Function Parameters

x	floating point value to use as the magnitude

Return Value

Returns the absolute value of `x`

Remarks

Domain: `-INF <= x <= INF`
Range: `0 <= y <= INF`
This function operates on double-precision floating point values, use SDL_copysignf for single-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_fabsf

SDL_fabsf

Compute the absolute value of `x`

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_fabsf(float x);
```

Function Parameters

x	floating point value to use as the magnitude

Return Value

Returns the absolute value of `x`

Remarks

Domain: `-INF <= x <= INF`

Range: `0 <= y <= INF`

This function operates on single-precision floating point values, use SDL_copysignf for double-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_fabs

SDL_FillSurfaceRect

Perform a fast fill of a rectangle with a specific color.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_FillSurfaceRect(SDL_Surface *dst, const SDL_Rect *rect, Uint32
    color);
```

Function Parameters

dst	the SDL_Surface structure that is the drawing target
rect	the SDL_Rect structure representing the rectangle to fill, or NULL to fill the entire surface
color	the color to fill with

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

`color` should be a pixel of the format used by the surface, and can be generated by SDL_MapRGB() or SDL_MapRGBA(). If the color value contains an alpha component then the destination is simply filled with that alpha information, no blending takes place.

If there is a clip rectangle set on the destination (set via SDL_SetSurfaceClipRect()), then this function will fill based on the intersection of the clip rectangle and `rect`.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_FillSurfaceRects

SDL_FillSurfaceRects

Perform a fast fill of a set of rectangles with a specific color.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_FillSurfaceRects(SDL_Surface *dst, const SDL_Rect *rects, int
    count, Uint32 color);
```

Function Parameters

dst	the SDL_Surface structure that is the drawing target
rects	an array of SDL_Rects representing the rectangles to fill.
count	the number of rectangles in the array
color	the color to fill with

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

`color` should be a pixel of the format used by the surface, and can be generated by SDL_MapRGB() or SDL_MapRGBA(). If the color value contains an alpha component then the destination is simply filled with that alpha information, no blending takes place.

If there is a clip rectangle set on the destination (set via SDL_SetSurfaceClipRect()), then this function will fill based on the intersection of the clip rectangle and `rect`.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_FillSurfaceRect

SDL_FilterEvents

Run a specific filter function on the current event queue, removing any events for which the filter returns 0.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
void SDL_FilterEvents(SDL_EventFilter filter, void *userdata);
```

Function Parameters

filter	the SDL_EventFilter function to call when an event happens
userdata	a pointer that is passed to `filter`

Remarks

See SDL_SetEventFilter() for more information. Unlike SDL_SetEventFilter(), this function does not change the filter permanently, it only uses the supplied filter until this function returns.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetEventFilter
- SDL_SetEventFilter

SDL_FlashWindow

Request a window to demand attention from the user.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_FlashWindow(SDL_Window *window, SDL_FlashOperation operation);
```

Function Parameters

window	the window to be flashed
operation	the flash operation

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_FlipSurface

Flip a surface vertically or horizontally.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_FlipSurface(SDL_Surface *surface, SDL_FlipMode flip);
```

Function Parameters

surface	the surface to flip
flip	the direction to flip

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_floor

Compute the floor of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_floor(double x);
```

Function Parameters

x	floating point value

Return Value

Returns the floor of `x`

Remarks

The floor of `x` is the largest integer `y` such that `y > x`, i.e `x` rounded down to the nearest integer.

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`, y integer

This function operates on double-precision floating point values, use SDL_floorf for single-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_floorf
- SDL_ceil
- SDL_trunc
- SDL_round
- SDL_lround

SDL_floorf

Compute the floor of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_floorf(float x);
```

Function Parameters

x	floating point value

Return Value

Returns the floor of `x`

Remarks

The floor of `x` is the largest integer `y` such that `y > x`, i.e `x` rounded down to the nearest integer.

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`, y integer

This function operates on single-precision floating point values, use SDL_floorf for double-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_floor
- SDL_ceilf
- SDL_truncf
- SDL_roundf
- SDL_lroundf

SDL_FlushAudioStream

Tell the stream that you're done sending data, and anything being buffered should be converted/resampled and made available immediately.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_FlushAudioStream(SDL_AudioStream *stream);
```

Function Parameters

stream	The audio stream to flush

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

It is legal to add more data to a stream after flushing, but there may be audio gaps in the output. Generally this is intended to signal the end of input, so the complete output becomes available.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PutAudioStreamData

SDL_FlushEvent

Clear events of a specific type from the event queue.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
void SDL_FlushEvent(Uint32 type);
```

Function Parameters

type	the type of event to be cleared; see SDL_EventType for details

Remarks

This will unconditionally remove any events from the queue that match `type`. If you need to remove a range of event types, use SDL_FlushEvents() instead.

It's also normal to just ignore events you don't care about in your event loop without calling this function.

This function only affects currently queued events. If you want to make sure that all pending OS events are flushed, you can call SDL_PumpEvents() on the main thread immediately before the flush call.

If you have user events with custom data that needs to be freed, you should use SDL_PeepEvents() to remove and clean up those events before calling this function.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_FlushEvents

SDL_FlushEvents

Clear events of a range of types from the event queue.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
void SDL_FlushEvents(Uint32 minType, Uint32 maxType);
```

Function Parameters

minType	the low end of event type to be cleared, inclusive; see SDL_EventType for details
maxType	the high end of event type to be cleared, inclusive; see SDL_EventType for details

Remarks

This will unconditionally remove any events from the queue that are in the range of `minType` to `maxType`, inclusive. If you need to remove a single event type, use SDL_FlushEvent() instead.

It's also normal to just ignore events you don't care about in your event loop without calling this function.

This function only affects currently queued events. If you want to make sure that all pending OS events are flushed, you can call SDL_PumpEvents() on the main thread immediately before the flush call.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_FlushEvent

SDL_FlushRenderer

Force the rendering context to flush any pending commands and state.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_FlushRenderer(SDL_Renderer *renderer);
```

Function Parameters

renderer	the rendering context

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

You do not need to (and in fact, shouldn't) call this function unless you are planning to call into OpenGL/Direct3D/Metal/whatever directly, in addition to using an SDL_Renderer.

This is for a very-specific case: if you are using SDL's render API, and you plan to make OpenGL/D3D/whatever calls in addition to SDL render API calls. If this applies, you should call this function between calls to SDL's render API and the low-level API you're using in cooperation.

In all other cases, you can ignore this function.

This call makes SDL flush any pending rendering work it was queueing up to do later in a single batch, and marks any internal cached state as invalid, so it'll prepare all its state again later, from scratch.

This means you do not need to save state in your rendering code to protect the SDL renderer. However, there lots of arbitrary pieces of Direct3D and OpenGL state that can confuse things; you should use your best judgement and be prepared to make changes if specific state needs to be protected.

Version

This function is available since SDL 3.0.0.

SDL_fmod

Return the floating-point remainder of `x / y`

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_fmod(double x, double y);
```

Function Parameters

x	the numerator
y	the denominator. Must not be 0.

Return Value

Returns the remainder of `x / y`

Remarks

Divides `x` by `y`, and returns the remainder.

Domain: `-INF <= x <= INF`, `-INF <= y <= INF`, `y != 0`

Range: `-y <= z <= y`

This function operates on double-precision floating point values, use SDL_fmodf for single-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_fmodf
- SDL_modf
- SDL_trunc
- SDL_ceil
- SDL_floor
- SDL_round
- SDL_lround

SDL_fmodf

Return the floating-point remainder of `x / y`

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_fmodf(float x, float y);
```

Function Parameters

x	the numerator
y	the denominator. Must not be 0.

Return Value

Returns the remainder of `x / y`

Remarks

Divides `x` by `y`, and returns the remainder.

Domain: `-INF <= x <= INF`, `-INF <= y <= INF`, `y != 0`

Range: `-y <= z <= y`

This function operates on single-precision floating point values, use SDL_fmod for single-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_fmod
- SDL_truncf
- SDL_modff
- SDL_ceilf
- SDL_floorf
- SDL_roundf
- SDL_lroundf

SDL_GamepadConnected

Check if a gamepad has been opened and is currently connected.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_bool SDL_GamepadConnected(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	a gamepad identifier previously returned by SDL_OpenGamepad()

Return Value

Returns SDL_TRUE if the gamepad has been opened and is currently connected, or SDL_FALSE if not.

Version

This function is available since SDL 3.0.0.

SDL_GamepadEventsEnabled

Query the state of gamepad event processing.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_bool SDL_GamepadEventsEnabled(void);
```

Return Value

Returns SDL_TRUE if gamepad events are being processed, SDL_FALSE otherwise.

Remarks

If gamepad events are disabled, you must call SDL_UpdateGamepads() yourself and check the state of the gamepad when you want gamepad information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetGamepadEventsEnabled

SDL_GamepadHasAxis

Query whether a gamepad has a given axis.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_bool SDL_GamepadHasAxis(SDL_Gamepad *gamepad, SDL_GamepadAxis axis);
```

Function Parameters

gamepad	a gamepad
axis	an axis enum value (an SDL_GamepadAxis value)

Return Value

Returns SDL_TRUE if the gamepad has this axis, SDL_FALSE otherwise.

Remarks

This merely reports whether the gamepad's mapping defined this axis, as that is all the information SDL has about the physical device.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GamepadHasButton
- SDL_GetGamepadAxis

SDL_GamepadHasButton

Query whether a gamepad has a given button.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_bool SDL_GamepadHasButton(SDL_Gamepad *gamepad, SDL_GamepadButton
    button);
```

Function Parameters

gamepad	a gamepad
button	a button enum value (an SDL_GamepadButton value)

Return Value

Returns SDL_TRUE if the gamepad has this button, SDL_FALSE otherwise.

Remarks

This merely reports whether the gamepad's mapping defined this button, as that is all the information SDL has about the physical device.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GamepadHasAxis

SDL_GamepadHasSensor

Return whether a gamepad has a particular sensor.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_bool SDL_GamepadHasSensor(SDL_Gamepad *gamepad, SDL_SensorType type);
```

Function Parameters

gamepad	The gamepad to query
type	The type of sensor to query

Return Value

Returns SDL_TRUE if the sensor exists, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadSensorData
- SDL_GetGamepadSensorDataRate
- SDL_SetGamepadSensorEnabled

SDL_GamepadSensorEnabled

Query whether sensor data reporting is enabled for a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_bool SDL_GamepadSensorEnabled(SDL_Gamepad *gamepad, SDL_SensorType
    type);
```

Function Parameters

gamepad	The gamepad to query
type	The type of sensor to query

Return Value

Returns SDL_TRUE if the sensor is enabled, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetGamepadSensorEnabled

SDL_GDKGetDefaultUser

Gets a reference to the default user handle for GDK.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_GDKGetDefaultUser(XUserHandle * outUserHandle);
```

Function Parameters

outUserHandle	a pointer to be filled in with the default user handle.

Return Value

Returns 0 if success, -1 if any error occurs.

Remarks

This is effectively a synchronous version of XUserAddAsync, which always prefers the default user and allows a sign-in UI.

Version

This function is available since SDL 3.0.0.

SDL_GDKGetTaskQueue

Gets a reference to the global async task queue handle for GDK, initializing if needed.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_GDKGetTaskQueue(XTaskQueueHandle * outTaskQueue);
```

Function Parameters

outTaskQueue	a pointer to be filled in with task queue handle.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Once you are done with the task queue, you should call XTaskQueueCloseHandle to reduce the reference count to avoid a resource leak.

Version

This function is available since SDL 3.0.0.

SDL_GDKSuspendComplete

Callback from the application to let the suspend continue.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
void SDL_GDKSuspendComplete(void);
```

Version

This function is available since SDL 3.0.0.

SDL_GetAndroidSDKVersion

Query Android API level of the current device.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_GetAndroidSDKVersion(void);
```

Return Value

Returns the Android API level.

Remarks

- API level 34: Android 14 (UPSIDE_DOWN_CAKE)
- API level 33: Android 13 (TIRAMISU)
- API level 32: Android 12L (S_V2)
- API level 31: Android 12 (S)
- API level 30: Android 11 (R)
- API level 29: Android 10 (Q)
- API level 28: Android 9 (P)
- API level 27: Android 8.1 (O_MR1)
- API level 26: Android 8.0 (O)
- API level 25: Android 7.1 (N_MR1)
- API level 24: Android 7.0 (N)
- API level 23: Android 6.0 (M)
- API level 22: Android 5.1 (LOLLIPOP_MR1)
- API level 21: Android 5.0 (LOLLIPOP, L)
- API level 20: Android 4.4W (KITKAT_WATCH)
- API level 19: Android 4.4 (KITKAT)

- API level 18: Android 4.3 (JELLY_BEAN_MR2)
- API level 17: Android 4.2 (JELLY_BEAN_MR1)
- API level 16: Android 4.1 (JELLY_BEAN)
- API level 15: Android 4.0.3 (ICE_CREAM_SANDWICH_MR1)
- API level 14: Android 4.0 (ICE_CREAM_SANDWICH)
- API level 13: Android 3.2 (HONEYCOMB_MR2)
- API level 12: Android 3.1 (HONEYCOMB_MR1)
- API level 11: Android 3.0 (HONEYCOMB)
- API level 10: Android 2.3.3 (GINGERBREAD_MR1)

Version

This function is available since SDL 3.0.0.

SDL_GetAssertionHandler

Get the current assertion handler.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
SDL_AssertionHandler SDL_GetAssertionHandler(void **puserdata);
```

Function Parameters

puserdata	pointer which is filled with the "userdata" pointer that was passed to SDL_SetAssertionHandler()

Return Value

Returns the SDL_AssertionHandler that is called when an assert triggers.

Remarks

This returns the function pointer that is called when an assertion is triggered. This is either the value last passed to SDL_SetAssertionHandler(), or if no application-specified function is set, is equivalent to calling SDL_GetDefaultAssertionHandler().

The parameter `puserdata` is a pointer to a void*, which will store the "userdata" pointer that was passed to SDL_SetAssertionHandler(). This value will always be NULL for the default handler. If you don't care about this data, it is safe to pass a NULL pointer to this function to ignore it.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetAssertionHandler

SDL_GetAssertionReport

Get a list of all assertion failures.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
const SDL_AssertData * SDL_GetAssertionReport(void);
```

Return Value

Returns a list of all failed assertions or NULL if the list is empty. This memory should not be modified or freed by the application.

Remarks

This function gets all assertions triggered since the last call to SDL_ResetAssertionReport(), or the start of the program.

The proper way to examine this data looks something like this:

```
const SDL_AssertData *item = SDL_GetAssertionReport();
while (item) {
   printf("'%s', %s (%s:%d), triggered %u times, always ignore: %s.\\n",
         item->condition, item->function, item->filename,
         item->linenum, item->trigger_count,
         item->always_ignore ? "yes" : "no");
   item = item->next;
}
```

Code Examples

The proper way to examine this data looks something like this:

```
const SDL_AssertData *item = SDL_GetAssertionReport();
while (item) {
     printf("'%s', %s (%s:%d), triggered %u times, always ignore: %s.\n",
         item->condition, item->function, item->filename,
         item->linenum, item->trigger_count,
         item->always_ignore ? "yes" : "no");
     item = item->next;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ResetAssertionReport

SDL_GetAudioCaptureDevices

Get a list of currently-connected audio capture devices.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
SDL_AudioDeviceID* SDL_GetAudioCaptureDevices(int *count);
```

Function Parameters

count	a pointer filled in with the number of devices returned

Return Value

Returns a 0 terminated array of device instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Remarks

This returns of list of available devices that record audio, like a microphone ("capture" devices). If you want devices that play sound, perhaps to speakers or headphones ("output" devices), use SDL_GetAudioOutputDevices() instead.

This only returns a list of physical devices; it will not have any device IDs returned by SDL_OpenAudioDevice().

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenAudioDevice
- SDL_GetAudioOutputDevices

SDL_GetAudioDeviceFormat

Get the current audio format of a specific audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_GetAudioDeviceFormat(SDL_AudioDeviceID devid, SDL_AudioSpec
    *spec, int *sample_frames);
```

Function Parameters

devid	the instance ID of the device to query.
spec	On return, will be filled with device details.
sample_frames	Pointer to store device buffer size, in sample frames. Can be NULL.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

For an opened device, this will report the format the device is currently using. If the device isn't yet opened, this will report the device's preferred format (or a reasonable default if this can't be determined).

You may also specify SDL_AUDIO_DEVICE_DEFAULT_OUTPUT or SDL_AUDIO_DEVICE_DEFAULT_CAPTU here, which is useful for getting a reasonable recommendation before opening the system-recommended default device.

You can also use this to request the current device buffer size. This is specified in sample frames and represents the amount of data SDL will feed to the physical hardware in each chunk. This can be converted to milliseconds of audio with the following equation: `ms = (int) ((((Sint64) frames) * 1000) / spec.freq);`

Buffer size is only important if you need low-level control over the audio playback timing. Most apps do not need this.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_GetAudioDeviceName

Get the human-readable name of a specific audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
char* SDL_GetAudioDeviceName(SDL_AudioDeviceID devid);
```

Function Parameters

devid	the instance ID of the device to query.

Return Value

Returns the name of the audio device, or NULL on error.

Remarks

The string returned by this function is UTF-8 encoded. The caller should call SDL_free on the return value when done with it.

Code Examples

```
int count;
SDL_AudioDeviceID *devices;
devices = SDL_GetAudioOutputDevices(&count);

for (int i = 0; i < count; ++i) {
    SDL_Log("Audio device %d: %s", i,
        SDL_GetAudioDeviceName(devices[i]));
}

SDL_free(devices);
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAudioOutputDevices
- SDL_GetAudioCaptureDevices
- SDL_GetDefaultAudioInfo

SDL_GetAudioDriver

Use this function to get the name of a built in audio driver.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
const char* SDL_GetAudioDriver(int index);
```

Function Parameters

index	the index of the audio driver; the value ranges from 0 to SDL_GetNumAudioDrivers() - 1

Return Value

Returns the name of the audio driver at the requested index, or NULL if an invalid index was specified.

Remarks

The list of audio drivers is given in the order that they are normally initialized by default; the drivers that seem more reasonable to choose first (as far as the SDL developers believe) are earlier in the list.

The names of drivers are all simple, low-ASCII identifiers, like "alsa", "coreaudio" or "xaudio2". These never have Unicode characters, and are not meant to be proper names.

Code Examples

```
int i;

for (i = 0; i < SDL_GetNumAudioDrivers(); ++i) {
    printf("Audio driver %d: %s\n", i, SDL_GetAudioDriver(i));
}
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumAudioDrivers

SDL_GetAudioOutputDevices

Get a list of currently-connected audio output devices.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
SDL_AudioDeviceID* SDL_GetAudioOutputDevices(int *count);
```

Function Parameters

count	a pointer filled in with the number of devices returned

Return Value

Returns a 0 terminated array of device instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Remarks

This returns of list of available devices that play sound, perhaps to speakers or headphones ("output" devices). If you want devices that record audio, like a microphone ("capture" devices), use SDL_GetAudioCaptureDevices() instead.

This only returns a list of physical devices; it will not have any device IDs returned by SDL_OpenAudioDevice().

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenAudioDevice
- SDL_GetAudioCaptureDevices

SDL_GetAudioStreamAvailable

Get the number of converted/resampled bytes available.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_GetAudioStreamAvailable(SDL_AudioStream *stream);
```

Function Parameters

stream	The audio stream to query

Return Value

Returns the number of converted/resampled bytes available.

Remarks

The stream may be buffering data behind the scenes until it has enough to resample correctly, so this number might be lower than what you expect, or even be zero. Add more data or flush the stream if you need the data now.

If the stream has so much data that it would overflow an int, the return value is clamped to a maximum value, but no queued data is lost; if there are gigabytes of data queued, the app might need to read some of it with SDL_GetAudioStreamData before this function's return value is no longer clamped.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAudioStreamData
- SDL_PutAudioStreamData

SDL_GetAudioStreamData

Get converted/resampled data from the stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_GetAudioStreamData(SDL_AudioStream *stream, void *buf, int len);
```

Function Parameters

stream	The stream the audio is being requested from
buf	A buffer to fill with audio data
len	The maximum number of bytes to fill

Return Value

Returns the number of bytes read from the stream, or -1 on error

Remarks

The input/output data format/channels/samplerate is specified when creating the stream, and can be changed after creation by calling SDL_SetAudioStreamFormat.

Note that any conversion and resampling necessary is done during this call, and SDL_PutAudioStreamData simply queues unconverted data for later. This is different than SDL2, where that work was done while inputting new data to the stream and requesting the output just copied the converted data.

Thread Safety

It is safe to call this function from any thread, but if the stream has a callback set, the caller might need to manage extra locking.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ClearAudioStream
- SDL_GetAudioStreamAvailable

- SDL_PutAudioStreamData

SDL_GetAudioStreamDevice

Query an audio stream for its currently-bound device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
SDL_AudioDeviceID SDL_GetAudioStreamDevice(SDL_AudioStream *stream);
```

Function Parameters

stream	the audio stream to query.

Return Value

Returns The bound audio device, or 0 if not bound or invalid.

Remarks

This reports the audio device that an audio stream is currently bound to.

If not bound, or invalid, this returns zero, which is not a valid device ID.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BindAudioStream
- SDL_BindAudioStreams

SDL_GetAudioStreamFormat

Query the current format of an audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_GetAudioStreamFormat(SDL_AudioStream *stream,
                             SDL_AudioSpec *src_spec,
                             SDL_AudioSpec *dst_spec);
```

Function Parameters

stream	the SDL_AudioStream to query.
src_spec	Where to store the input audio format; ignored if NULL.
dst_spec	Where to store the output audio format; ignored if NULL.

Return Value

Returns 0 on success, or -1 on error.

Thread Safety

It is safe to call this function from any thread, as it holds a stream-specific mutex while running.

Version

This function is available since SDL 3.0.0.

SDL_GetAudioStreamFrequencyRatio

Get the frequency ratio of an audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
float SDL_GetAudioStreamFrequencyRatio(SDL_AudioStream *stream);
```

Function Parameters

stream	the SDL_AudioStream to query.

Return Value

Returns the frequency ratio of the stream, or 0.0 on error

Thread Safety

It is safe to call this function from any thread, as it holds a stream-specific mutex while running.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetAudioStreamFrequencyRatio

SDL_GetAudioStreamProperties

Get the properties associated with an audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
SDL_PropertiesID SDL_GetAudioStreamProperties(SDL_AudioStream *stream);
```

Function Parameters

stream	the SDL_AudioStream to query

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetAudioStreamQueued

Get the number of bytes currently queued.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_GetAudioStreamQueued(SDL_AudioStream *stream);
```

Function Parameters

stream	The audio stream to query

Return Value

Returns the number of bytes queued.

Remarks

Note that audio streams can change their input format at any time, even if there is still data queued in a different format, so the returned byte count will not necessarily match the number of _sample frames_ available. Users of this API should be aware of format changes they make when feeding a stream and plan accordingly.

Queued data is not converted until it is consumed by SDL_GetAudioStreamData, so this value should be representative of the exact data that was put into the stream.

If the stream has so much data that it would overflow an int, the return value is clamped to a maximum value, but no queued data is lost; if there are gigabytes of data queued, the app might need to read some of it with SDL_GetAudioStreamData before this function's return value is no longer clamped.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PutAudioStreamData
- SDL_ClearAudioStream

SDL_GetBasePath

Get the directory where the application was run from.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
char* SDL_GetBasePath(void);
```

Return Value

Returns an absolute path in UTF-8 encoding to the application data directory. NULL will be returned on error or when the platform doesn't implement this functionality, call SDL_GetError() for more information.

Remarks

This is not necessarily a fast call, so you should call this once near startup and save the string if you need it. **macOS and iOS Specific Functionality**: If the application is in a ".app" bundle, this function returns the Resource directory (e.g. MyApp.app/Contents/Resources/). This behaviour can be overridden by adding a property to the Info.plist file. Adding a string key with the name SDL_FILESYSTEM_BASE_DIR_TYPE with a supported value will change the behaviour.

Supported values for the SDL_FILESYSTEM_BASE_DIR_TYPE property (Given an application in /Applications/SDLApp/MyApp.app):

- `resource`: bundle resource directory (the default). For example: `/Applications/SDLApp/MyApp.app/Contents`
- `bundle`: the Bundle directory. For example: `/Applications/SDLApp/MyApp.app/`
- `parent`: the containing directory of the bundle. For example: `/Applications/SDLApp/`
  Nintendo 3DS Specific Functionality: This function returns "romfs" directory of the application as it is uncommon to store resources outside the executable. As such it is not a writable directory.

The returned path is guaranteed to end with a path separator ('
' on Windows, '/' on most other platforms).

The pointer returned is owned by the caller. Please call SDL_free() on the pointer when done with it.

Code Examples

```
char *data_path = NULL;

void InitializeDataPath(void) {
    char *base_path = SDL_GetBasePath();
    if (base_path) {
        data_path = base_path;
    } else {
        data_path = SDL_strdup("./");
    }
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPrefPath

SDL_GetBooleanProperty

Get a boolean property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
SDL_bool SDL_GetBooleanProperty(SDL_PropertiesID props, const char
    *name, SDL_bool default_value);
```

Function Parameters

props	the properties to query
name	the name of the property to query
default_value	the default value of the property

Return Value

Returns the value of the property, or `default_value` if it is not set or not a float property.

Remarks

You can use SDL_GetPropertyType() to query whether the property exists and is a boolean property.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPropertyType
- SDL_HasProperty
- SDL_SetBooleanProperty

SDL_GetCameraDeviceName

Get human-readable device name for a camera.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
char * SDL_GetCameraDeviceName(SDL_CameraDeviceID instance_id);
```

Function Parameters

instance_id	the camera device instance ID

Return Value

Returns Human-readable device name, or NULL on error; call SDL_GetError() for more information.

Remarks

The returned string is owned by the caller; please release it with SDL_free() when done with it.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetCameraDevices

SDL_GetCameraDevicePosition

Get the position of the camera in relation to the system.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
SDL_CameraPosition SDL_GetCameraDevicePosition(SDL_CameraDeviceID
    instance_id);
```

Function Parameters

instance_id	the camera device instance ID

Return Value

Returns The position of the camera on the system hardware.

Remarks

Most platforms will report UNKNOWN, but mobile devices, like phones, can often make a distiction between cameras on the front of the device (that points towards the user, for taking "selfies") and cameras on the back (for filming in the direction the user is facing).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetCameraDevices

SDL_GetCameraDevices

Get a list of currently connected camera devices.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
SDL_CameraDeviceID* SDL_GetCameraDevices(int *count);
```

Function Parameters

count	a pointer filled in with the number of camera devices. Can be NULL.

Return Value

Returns a 0 terminated array of camera instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenCamera

SDL_GetCameraDeviceSupportedFormats

Get the list of native formats/sizes a camera supports.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
SDL_CameraSpec* SDL_GetCameraDeviceSupportedFormats(SDL_CameraDeviceID
    devid, int *count);
```

Function Parameters

devid	the camera device instance ID to query.
count	a pointer filled in with the number of elements in the list. Can be NULL.

Return Value

Returns a 0 terminated array of SDL_CameraSpecs, which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Remarks

This returns a list of all formats and frame sizes that a specific camera can offer. This is useful if your app can accept a variety of image formats and sizes and so want to find the optimal spec that doesn't require conversion.

This function isn't strictly required; if you call SDL_OpenCameraDevice with a NULL spec, SDL will choose a native format for you, and if you instead specify a desired format, it will transparently convert to the requested format on your behalf.

If `count` is not NULL, it will be filled with the number of elements in the returned array. Additionally, the last element of the array has all fields set to zero (this element is not included in `count`).

The returned list is owned by the caller, and should be released with SDL_free() when no longer needed.

Note that it's legal for a camera to supply a list with only the zeroed final element and `*count` set to zero; this is what will happen on Emscripten builds, since that platform won't tell _anything_ about available cameras until you've opened one, and won't even tell if there _is_ a camera until the user has given you permission to check through a scary warning popup.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetCameraDevices
- SDL_OpenCameraDevice

SDL_GetCameraDriver

Use this function to get the name of a built in camera driver.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
const char* SDL_GetCameraDriver(int index);
```

Function Parameters

index	the index of the camera driver; the value ranges from 0 to SDL_GetNumCameraDrivers() - 1

Return Value

Returns the name of the camera driver at the requested index, or NULL if an invalid index was specified.

Remarks

The list of camera drivers is given in the order that they are normally initialized by default; the drivers that seem more reasonable to choose first (as far as the SDL developers believe) are earlier in the list.

The names of drivers are all simple, low-ASCII identifiers, like "v4l2", "coremedia" or "android". These never have Unicode characters, and are not meant to be proper names.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumCameraDrivers

SDL_GetCameraFormat

Get the spec that a camera is using when generating images.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
int SDL_GetCameraFormat(SDL_Camera *camera, SDL_CameraSpec *spec);
```

Function Parameters

camera	opened camera device
spec	The SDL_CameraSpec to be initialized by this function.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Note that this might not be the native format of the hardware, as SDL might be converting to this format behind the scenes.

If the system is waiting for the user to approve access to the camera, as some platforms require, this will return -1, but this isn't necessarily a fatal error; you should either wait for an SDL_EVENT_CAMERA_DEVICE_APPROVED (or SDL_EVENT_CAMERA_DEVICE_DENIED) event, or poll SDL_IsCameraApproved() occasionally until it returns non-zero.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenCameraDevice

SDL_GetCameraInstanceID

Get the instance ID of an opened camera.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
SDL_CameraDeviceID SDL_GetCameraInstanceID(SDL_Camera *camera);
```

Function Parameters

camera	an SDL_Camera to query

Return Value

Returns the instance ID of the specified camera on success or 0 on failure; call SDL_GetError() for more information.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenCameraDevice

SDL_GetCameraPermissionState

Query if camera access has been approved by the user.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
int SDL_GetCameraPermissionState(SDL_Camera *camera);
```

Function Parameters

camera	the opened camera device to query

Return Value

Returns -1 if user denied access to the camera, 1 if user approved access, 0 if no decision has been made yet.

Remarks

Cameras will not function between when the device is opened by the app and when the user permits access to the hardware. On some platforms, this presents as a popup dialog where the user has to explicitly approve access; on others the approval might be implicit and not alert the user at all.

This function can be used to check the status of that approval. It will return 0 if still waiting for user response, 1 if the camera is approved for use, and -1 if the user denied access.

Instead of polling with this function, you can wait for a SDL_EVENT_CAMERA_DEVICE_APPROVE (or SDL_EVENT_CAMERA_DEVICE_DENIED) event in the standard SDL event loop, which is guaranteed to be sent once when permission to use the camera is decided.

If a camera is declined, there's nothing to be done but call SDL_CloseCamera() to dispose of it.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenCameraDevice
- SDL_CloseCamera

SDL_GetCameraProperties

Get the properties associated with an opened camera.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
SDL_PropertiesID SDL_GetCameraProperties(SDL_Camera *camera);
```

Function Parameters

camera	the SDL_Camera obtained from SDL_OpenCameraDevice()

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetClipboardData

Get the data from clipboard for a given mime type.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
void* SDL_GetClipboardData(const char *mime_type, size_t *size);
```

Function Parameters

mime_type	The mime type to read from the clipboard
size	A pointer filled in with the length of the returned data

Return Value

Returns the retrieved data buffer or NULL on failure; call SDL_GetError() for more information. Caller must call SDL_free() on the returned pointer when done with it.

Remarks

The size of text data does not include the terminator, but the text is guaranteed to be null terminated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasClipboardData
- SDL_SetClipboardData

SDL_GetClipboardText

Get UTF-8 text from the clipboard, which must be freed with SDL_free().

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
char * SDL_GetClipboardText(void);
```

Return Value

Returns the clipboard text on success or an empty string on failure; call SDL_GetError() for more information. Caller must call SDL_free() on the returned pointer when done with it (even if there was an error).

Remarks

This functions returns empty string if there was not enough memory left for a copy of the clipboard's content.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasClipboardText
- SDL_SetClipboardText

SDL_GetClosestFullscreenDisplayMode

Get the closest match to the requested display mode.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
const SDL_DisplayMode* SDL_GetClosestFullscreenDisplayMode(SDL_DisplayID
    displayID, int w, int h, float refresh_rate, SDL_bool
    include_high_density_modes);
```

Function Parameters

displayID	the instance ID of the display to query
w	the width in pixels of the desired display mode
h	the height in pixels of the desired display mode
refresh_rate	the refresh rate of the desired display mode, or 0.0f for the desktop refresh rate
include_high_density_modes	Boolean to include high density modes in the search

Return Value

Returns a pointer to the closest display mode equal to or larger than the desired mode, or NULL on error; call SDL_GetError() for more information.

Remarks

The available display modes are scanned and `closest` is filled in with the closest mode matching the requested mode and returned. The mode format and refresh rate default to the desktop mode if they are set to 0. The modes are scanned with size being first priority, format being second priority, and finally checking the refresh rate. If all the available modes are too small, then NULL is returned.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplays
- SDL_GetFullscreenDisplayModes

SDL_GetCPUCacheLineSize

Determine the L1 cache line size of the CPU.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
int SDL_GetCPUCacheLineSize(void);
```

Return Value

Returns the L1 cache line size of the CPU, in bytes.

Remarks

This is useful for determining multi-threaded structure padding or SIMD prefetch sizes.

Version

This function is available since SDL 3.0.0.

SDL_GetCPUCount

Get the number of CPU cores available.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
int SDL_GetCPUCount(void);
```

Return Value

Returns the total number of logical CPU cores. On CPUs that include technologies such as hyperthreading, the number of logical cores may be more than the number of physical cores.

Code Examples

```
SDL_Log("Number of logical CPU cores: %d", SDL_GetCPUCount());
```

Version

This function is available since SDL 3.0.0.

SDL_GetCurrentAudioDriver

Get the name of the current audio driver.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
const char* SDL_GetCurrentAudioDriver(void);
```

Return Value

Returns the name of the current audio driver or NULL if no driver has been initialized.

Remarks

The returned string points to internal static memory and thus never becomes invalid, even if you quit the audio subsystem and initialize a new driver (although such a case would return a different static string from another call to this function, of course). As such, you should not modify or free the returned string.

Code Examples

```
const char* driver_name = SDL_GetCurrentAudioDriver();

if (driver_name) {
    printf("Audio subsystem initialized; driver = %s.\n", driver_name);
} else {
    printf("Audio subsystem not initialized.\n");
}
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_GetCurrentCameraDriver

Get the name of the current camera driver.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
const char* SDL_GetCurrentCameraDriver(void);
```

Return Value

Returns the name of the current camera driver or NULL if no driver has been initialized.

Remarks

The returned string points to internal static memory and thus never becomes invalid, even if you quit the camera subsystem and initialize a new driver (although such a case would return a different static string from another call to this function, of course). As such, you should not modify or free the returned string.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_GetCurrentDisplayMode

Get information about the current display mode.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
const SDL_DisplayMode* SDL_GetCurrentDisplayMode(SDL_DisplayID
    displayID);
```

Function Parameters

displayID	the instance ID of the display to query

Return Value

Returns a pointer to the desktop display mode or NULL on error; call SDL_GetError() for more information.

Remarks

There's a difference between this function and SDL_GetDesktopDisplayMode() when SDL runs fullscreen and has changed the resolution. In that case this function will return the current display mode, and not the previous native display mode.

Code Examples

```
// Using SDL2's SDL_GetCurrentDisplayMode()

#include <SDL3/SDL.h>

int main(int argc, char* argv[])
{
  int i;

  // Declare display mode structure to be filled in.
  SDL_DisplayMode current;

  SDL_Init(SDL_INIT_VIDEO);

  int count_displays;
```

```
  SDL_DisplayID *displays = SDL_GetDisplays(&count_displays);

  // Get current display mode of all displays.
  for(i = 0; i < count_displays; i++){

    const SDL_DisplayMode *display_mode =
        SDL_GetCurrentDisplayMode(displays[i]);

    if (display_mode== NULL) {
      // In case of error...
      SDL_Log("Could not get display mode for video display #%d: %s", i,
          SDL_GetError());
    } else {
      // On success, print the current display mode.
      SDL_Log("Display #%d: current display mode is %dx%dpx @ %dhz.", i,
          display_mode->w, display_mode->h, display_mode->refresh_rate);
    }
  }

  // Clean up and exit the program.
  SDL_free(displays);
  SDL_Quit();
  return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDesktopDisplayMode
- SDL_GetDisplays

SDL_GetCurrentDisplayOrientation

Get the orientation of a display.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_DisplayOrientation SDL_GetCurrentDisplayOrientation(SDL_DisplayID
    displayID);
```

Function Parameters

displayID	the instance ID of the display to query

Return Value

Returns The SDL_DisplayOrientation enum value of the display, or `SDL_ORIENTATION_UNKNOWN` if it isn't available.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplays

SDL_GetCurrentRenderOutputSize

Get the current output size in pixels of a rendering context.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetCurrentRenderOutputSize(SDL_Renderer *renderer, int *w, int
    *h);
```

Function Parameters

renderer	the rendering context
w	a pointer filled in with the current width
h	a pointer filled in with the current height

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If a rendering target is active, this will return the size of the rendering target in pixels, otherwise if a logical size is set, it will return the logical size, otherwise it will return the value of SDL_GetRenderOutputSize().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderOutputSize

SDL_GetCurrentThreadID

Get the thread identifier for the current thread.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
SDL_ThreadID SDL_GetCurrentThreadID(void);
```

Return Value

Returns the ID of the current thread.

Remarks

This thread identifier is as reported by the underlying operating system. If SDL is running on a platform that does not support threads the return value will always be zero.

This function also returns a valid thread ID when called from the main thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetThreadID

SDL_GetCurrentTime

Gets the current value of the system realtime clock in nanoseconds since Jan 1, 1970 in Universal Coordinated Time (UTC).

Header File

Defined in SDL3/SDL_time.h

Syntax

```
int SDL_GetCurrentTime(SDL_Time *ticks);
```

Function Parameters

ticks	the SDL_Time to hold the returned tick count

Return Value

Returns 0 on success or -1 on error; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetCurrentVideoDriver

Get the name of the currently initialized video driver.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
const char* SDL_GetCurrentVideoDriver(void);
```

Return Value

Returns the name of the current video driver or NULL if no driver has been initialized.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumVideoDrivers
- SDL_GetVideoDriver

SDL_GetCursor

Get the active cursor.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_Cursor* SDL_GetCursor(void);
```

Return Value

Returns the active cursor or NULL if there is no mouse.

Remarks

This function returns a pointer to the current cursor which is owned by the library. It is not necessary to free the cursor with SDL_DestroyCursor().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetCursor

SDL_GetDayOfWeek

Get the day of week for a calendar date.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
int SDL_GetDayOfWeek(int year, int month, int day);
```

Function Parameters

year	the year component of the date
month	the month component of the date
day	the day component of the date

Return Value

Returns a value between 0 and 6 (0 being Sunday) if the date is valid, otherwise -1; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetDayOfYear

Get the day of year for a calendar date.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
int SDL_GetDayOfYear(int year, int month, int day);
```

Function Parameters

year	the year component of the date
month	the month component of the date
day	the day component of the date

Return Value

Returns the day of year [0-365] if the date is valid, otherwise -1; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetDaysInMonth

Get the number of days in a month for a given year.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
int SDL_GetDaysInMonth(int year, int month);
```

Function Parameters

year	the year
month	the month [1-12]

Return Value

Returns the number of days in the requested month, otherwise -1; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetDefaultAssertionHandler

Get the default assertion handler.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
SDL_AssertionHandler SDL_GetDefaultAssertionHandler(void);
```

Return Value

Returns the default SDL_AssertionHandler that is called when an assert triggers.

Remarks

This returns the function pointer that is called by default when an assertion is triggered. This is an internal function provided by SDL, that is used for assertions when SDL_SetAssertionHandler() hasn't been used to provide a different function.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAssertionHandler

SDL_GetDefaultCursor

Get the default cursor.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_Cursor* SDL_GetDefaultCursor(void);
```

Return Value

Returns the default cursor on success or NULL on failure.

Remarks

You do not have to call SDL_DestroyCursor() on the return value, but it is safe to do so.

Version

This function is available since SDL 3.0.0.

SDL_GetDesktopDisplayMode

Get information about the desktop's display mode.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
const SDL_DisplayMode* SDL_GetDesktopDisplayMode(SDL_DisplayID
    displayID);
```

Function Parameters

displayID	the instance ID of the display to query

Return Value

Returns a pointer to the desktop display mode or NULL on error; call SDL_GetError() for more information.

Remarks

There's a difference between this function and SDL_GetCurrentDisplayMode() when SDL runs fullscreen and has changed the resolution. In that case this function will return the previous native display mode, and not the current display mode.

Code Examples

```
SDL_DisplayID display_id = SDL_GetPrimaryDisplay();
const SDL_DisplayMode *display_mode =
    SDL_GetDesktopDisplayMode(display_id);
if (display_mode == NULL) {
    SDL_Log("SDL_GetDesktopDisplayMode failed: %s", SDL_GetError());
    return 1;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetCurrentDisplayMode
- SDL_GetDisplays

SDL_GetDisplayBounds

Get the desktop area represented by a display.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetDisplayBounds(SDL_DisplayID displayID, SDL_Rect *rect);
```

Function Parameters

displayID	the instance ID of the display to query
rect	the SDL_Rect structure filled in with the display bounds

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The primary display is always located at (0,0).

Code Examples

```
SDL_Rect r;
if (SDL_GetDisplayBounds(0, &r) != 0) {
    SDL_Log("SDL_GetDisplayBounds failed: %s", SDL_GetError());
    return 1;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplayUsableBounds
- SDL_GetDisplays

SDL_GetDisplayContentScale

Get the content scale of a display.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
float SDL_GetDisplayContentScale(SDL_DisplayID displayID);
```

Function Parameters

displayID	the instance ID of the display to query

Return Value

Returns The content scale of the display, or 0.0f on error; call SDL_GetError() for more details.

Remarks

The content scale is the expected scale for content based on the DPI settings of the display. For example, a 4K display might have a 2.0 (200display scale, which means that the user expects UI elements to be twice as big on this display, to aid in readability.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplays

SDL_GetDisplayForPoint

Get the display containing a point.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_DisplayID SDL_GetDisplayForPoint(const SDL_Point *point);
```

Function Parameters

point	the point to query

Return Value

Returns the instance ID of the display containing the point or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplayBounds
- SDL_GetDisplays

SDL_GetDisplayForRect

Get the display primarily containing a rect.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_DisplayID SDL_GetDisplayForRect(const SDL_Rect *rect);
```

Function Parameters

rect	the rect to query

Return Value

Returns the instance ID of the display entirely containing the rect or closest to the center of the rect on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplayBounds
- SDL_GetDisplays

SDL_GetDisplayForWindow

Get the display associated with a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_DisplayID SDL_GetDisplayForWindow(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the instance ID of the display containing the center of the window on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplayBounds
- SDL_GetDisplays

SDL_GetDisplayName

Get the name of a display in UTF-8 encoding.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
const char* SDL_GetDisplayName(SDL_DisplayID displayID);
```

Function Parameters

displayID	the instance ID of the display to query

Return Value

Returns the name of a display or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplays

SDL_GetDisplayProperties

Get the properties associated with a display.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_PropertiesID SDL_GetDisplayProperties(SDL_DisplayID displayID);
```

Function Parameters

displayID	the instance ID of the display to query

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Remarks

The following read-only properties are provided by SDL:

- `SDL_PROP_DISPLAY_HDR_ENABLED_BOOLEAN`: true if the display has HDR headroom above the SDR white point. This property can change dynamically when SDL_EVENT_DISPLAY_HDR_STATE_CHANGED is sent.
- `SDL_PROP_DISPLAY_SDR_WHITE_POINT_FLOAT`: the value of SDR white in the SDL_COLORSPACE_SRGB_LINEAR colorspace. On Windows this corresponds to the SDR white level in scRGB colorspace, and on Apple platforms this is always 1.0 for EDR content. This property can change dynamically when SDL_EVENT_DISPLAY_HDR_STATE_CHANGED is sent.
- `SDL_PROP_DISPLAY_HDR_HEADROOM_FLOAT`: the additional high dynamic range that can be displayed, in terms of the SDR white point. When HDR is not enabled, this will be 1.0. This property can change dynamically when SDL_EVENT_DISPLAY_HDR_STATE_CHANGED is sent.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetDisplays

Get a list of currently connected displays.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_DisplayID* SDL_GetDisplays(int *count);
```

Function Parameters

count	a pointer filled in with the number of displays returned

Return Value

Returns a 0 terminated array of display instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Version

This function is available since SDL 3.0.0.

SDL_GetDisplayUsableBounds

Get the usable desktop area represented by a display, in screen coordinates.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetDisplayUsableBounds(SDL_DisplayID displayID, SDL_Rect *rect);
```

Function Parameters

displayID	the instance ID of the display to query
rect	the SDL_Rect structure filled in with the display bounds

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This is the same area as SDL_GetDisplayBounds() reports, but with portions reserved by the system removed. For example, on Apple's macOS, this subtracts the area occupied by the menu bar and dock.

Setting a window to be fullscreen generally bypasses these unusable areas, so these are good guidelines for the maximum space available to a non-fullscreen window.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplayBounds
- SDL_GetDisplays

SDL_GetError

Retrieve a message about the last error that occurred on the current thread.

Header File

Defined in SDL3/SDL_error.h

Syntax

```
const char* SDL_GetError(void);
```

Return Value

Returns a message with information about the specific error that occurred, or an empty string if there hasn't been an error message set since the last call to SDL_ClearError(). The message is only applicable when an SDL function has signaled an error. You must check the return values of SDL function calls to determine when to appropriately call SDL_GetError().

Remarks

It is possible for multiple errors to occur before calling SDL_GetError(). Only the last error is returned.

The message is only applicable when an SDL function has signaled an error. You must check the return values of SDL function calls to determine when to appropriately call SDL_GetError(). You should *not* use the results of SDL_GetError() to decide if an error has occurred! Sometimes SDL will set an error string even when reporting success.

SDL will *not* clear the error string for successful API calls. You *must* check return values for failure cases before you can assume the error string applies.

Error strings are set per-thread, so an error set in a different thread will not interfere with the current thread's operation.

The returned string is internally allocated and must not be freed by the application.

Code Examples

```
if (SDL_Init(SDL_INIT_VIDEO) < 0) {
    // Unrecoverable error, exit here.
    printf("SDL_Init failed: %s\n", SDL_GetError());
}
```

Note: Although this example uses SDL_Init(), SDL_GetError() provides an error message for any failed SDL operation which supports error reporting, see the wiki page for each particular SDL function.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ClearError
- SDL_SetError

SDL_GetEventFilter

Query the current event filter.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
SDL_bool SDL_GetEventFilter(SDL_EventFilter *filter, void **userdata);
```

Function Parameters

filter	the current callback function will be stored here
userdata	the pointer that is passed to the current event filter will be stored here

Return Value

Returns SDL_TRUE on success or SDL_FALSE if there is no event filter set.

Remarks

This function can be used to "chain" filters, by saving the existing filter before replacing it with a function that will call that saved filter.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetEventFilter

SDL_GetFloatProperty

Get a floating point property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
float SDL_GetFloatProperty(SDL_PropertiesID props, const char *name,
    float default_value);
```

Function Parameters

props	the properties to query
name	the name of the property to query
default_value	the default value of the property

Return Value

Returns the value of the property, or `default_value` if it is not set or not a float property.

Remarks

You can use SDL_GetPropertyType() to query whether the property exists and is a floating point property.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPropertyType
- SDL_HasProperty
- SDL_SetFloatProperty

SDL_GetFullscreenDisplayModes

Get a list of fullscreen display modes available on a display.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
extern DECLSPEC const SDL_DisplayMode **SDLCALL
    SDL_GetFullscreenDisplayModes(SDL_DisplayID displayID, int *count);
```

Function Parameters

displayID	the instance ID of the display to query
count	a pointer filled in with the number of display modes returned

Return Value

Returns a NULL terminated array of display mode pointers which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Remarks

The display modes are sorted in this priority:

- w -¿ largest to smallest
- h -¿ largest to smallest
- bits per pixel -¿ more colors to fewer colors
- packed pixel layout -¿ largest to smallest
- refresh rate -¿ highest to lowest
- pixel density -¿ lowest to highest

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplays

SDL_GetGamepadAppleSFSymbolsNameForAxis

Return the sfSymbolsName for a given axis on a gamepad on Apple platforms.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char* SDL_GetGamepadAppleSFSymbolsNameForAxis(SDL_Gamepad
    *gamepad, SDL_GamepadAxis axis);
```

Function Parameters

gamepad	the gamepad to query
axis	an axis on the gamepad

Return Value

Returns the sfSymbolsName or NULL if the name can't be found

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadAppleSFSymbolsNameForButton

SDL_GetGamepadAppleSFSymbolsNameForButton

Return the sfSymbolsName for a given button on a gamepad on Apple platforms.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char* SDL_GetGamepadAppleSFSymbolsNameForButton(SDL_Gamepad
    *gamepad, SDL_GamepadButton button);
```

Function Parameters

gamepad	the gamepad to query
button	a button on the gamepad

Return Value

Returns the sfSymbolsName or NULL if the name can't be found

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadAppleSFSymbolsNameForAxis

SDL_GetGamepadAxis

Get the current state of an axis control on a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Sint16 SDL_GetGamepadAxis(SDL_Gamepad *gamepad, SDL_GamepadAxis axis);
```

Function Parameters

gamepad	a gamepad
axis	an axis index (one of the SDL_GamepadAxis values)

Return Value

Returns axis state (including 0) on success or 0 (also) on failure; call SDL_GetError() for more information.

Remarks

The axis indices start at index 0.

For thumbsticks, the state is a value ranging from -32768 (up/left) to 32767 (down/right).

Triggers range from 0 when released to 32767 when fully pressed, and never return a negative value. Note that this differs from the value reported by the lower-level SDL_GetJoystickAxis(), which normally uses the full range.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GamepadHasAxis
- SDL_GetGamepadButton

SDL_GetGamepadAxisFromString

Convert a string into SDL_GamepadAxis enum.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_GamepadAxis SDL_GetGamepadAxisFromString(const char *str);
```

Function Parameters

str	string representing a SDL_Gamepad axis

Return Value

Returns the SDL_GamepadAxis enum corresponding to the input string, or `SDL_GAMEPAD_AXIS_INVALID` if no match was found.

Remarks

This function is called internally to translate SDL_Gamepad mapping strings for the underlying joystick device into the consistent SDL_Gamepad mapping. You do not normally need to call this function unless you are parsing SDL_Gamepad mappings in your own code.

Note specially that "righttrigger" and "lefttrigger" map to `SDL_GAMEPAD_AXIS_RIGHT_TRIGGER` and `SDL_GAMEPAD_AXIS_LEFT_TRIGGER`, respectively.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadStringForAxis

SDL_GetGamepadBindings

Get the SDL joystick layer bindings for a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
extern DECLSPEC SDL_GamepadBinding **SDLCALL
    SDL_GetGamepadBindings(SDL_Gamepad *gamepad, int *count);
```

Function Parameters

gamepad	a gamepad
count	a pointer filled in with the number of bindings returned

Return Value

Returns a NULL terminated array of pointers to bindings which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadButton

Get the current state of a button on a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Uint8 SDL_GetGamepadButton(SDL_Gamepad *gamepad, SDL_GamepadButton
    button);
```

Function Parameters

gamepad	a gamepad
button	a button index (one of the SDL_GamepadButton values)

Return Value

Returns 1 for pressed state or 0 for not pressed state or error; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GamepadHasButton
- SDL_GetGamepadAxis

SDL_GetGamepadButtonFromString

Convert a string into an SDL_GamepadButton enum.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_GamepadButton SDL_GetGamepadButtonFromString(const char *str);
```

Function Parameters

str	string representing a SDL_Gamepad axis

Return Value

Returns the SDL_GamepadButton enum corresponding to the input string, or `SDL_GAMEPAD_BUTTON_INVALID` if no match was found.

Remarks

This function is called internally to translate SDL_Gamepad mapping strings for the underlying joystick device into the consistent SDL_Gamepad mapping. You do not normally need to call this function unless you are parsing SDL_Gamepad mappings in your own code.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadStringForButton

SDL_GetGamepadButtonLabel

Get the label of a button on a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_GamepadButtonLabel SDL_GetGamepadButtonLabel(SDL_Gamepad *gamepad,
    SDL_GamepadButton button);
```

Function Parameters

gamepad	a gamepad
button	a button index (one of the SDL_GamepadButton values)

Return Value

Returns the SDL_GamepadButtonLabel enum corresponding to the button label

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadButtonLabelForType

SDL_GetGamepadButtonLabelForType

Get the label of a button on a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_GamepadButtonLabel SDL_GetGamepadButtonLabelForType(SDL_GamepadType
    type, SDL_GamepadButton button);
```

Function Parameters

type	the type of gamepad to check
button	a button index (one of the SDL_GamepadButton values)

Return Value

Returns the SDL_GamepadButtonLabel enum corresponding to the button label

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadButtonLabel

SDL_GetGamepadConnectionState

Get the connection state of a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_JoystickConnectionState SDL_GetGamepadConnectionState(SDL_Gamepad
    *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the connection state on success or `SDL_JOYSTICK_CONNECTION_INVALID` on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadFirmwareVersion

Get the firmware version of an opened gamepad, if available.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Uint16 SDL_GetGamepadFirmwareVersion(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the gamepad firmware version, or zero if unavailable.

Remarks

If the firmware version isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadFromInstanceID

Get the SDL_Gamepad associated with a joystick instance ID, if it has been opened.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_Gamepad* SDL_GetGamepadFromInstanceID(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID of the gamepad

Return Value

Returns an SDL_Gamepad on success or NULL on failure or if it hasn't been opened yet; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadFromPlayerIndex

Get the SDL_Gamepad associated with a player index.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_Gamepad* SDL_GetGamepadFromPlayerIndex(int player_index);
```

Function Parameters

player_index	the player index, which different from the instance ID

Return Value

Returns the SDL_Gamepad associated with a player index.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadPlayerIndex
- SDL_SetGamepadPlayerIndex

SDL_GetGamepadInstanceGUID

Get the implementation-dependent GUID of a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_JoystickGUID SDL_GetGamepadInstanceGUID(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the GUID of the selected gamepad. If called on an invalid index, this function returns a zero GUID

Remarks

This can be called before any gamepads are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadGUID
- SDL_GetGamepadGUIDString
- SDL_GetGamepads

SDL_GetGamepadInstanceID

Get the instance ID of an opened gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_JoystickID SDL_GetGamepadInstanceID(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	a gamepad identifier previously returned by SDL_OpenGamepad()

Return Value

Returns the instance ID of the specified gamepad on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadInstanceMapping

Get the mapping of a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
char* SDL_GetGamepadInstanceMapping(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the mapping string. Must be freed with SDL_free(). Returns NULL if no mapping is available.

Remarks

This can be called before any gamepads are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepads
- SDL_GetGamepadMapping

SDL_GetGamepadInstanceName

Get the implementation dependent name of a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char* SDL_GetGamepadInstanceName(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the name of the selected gamepad. If no name can be found, this function returns NULL; call SDL_GetError() for more information.

Remarks

This can be called before any gamepads are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadName
- SDL_GetGamepads

SDL_GetGamepadInstancePath

Get the implementation dependent path of a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char* SDL_GetGamepadInstancePath(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the path of the selected gamepad. If no path can be found, this function returns NULL; call SDL_GetError() for more information.

Remarks

This can be called before any gamepads are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadPath
- SDL_GetGamepads

SDL_GetGamepadInstancePlayerIndex

Get the player index of a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_GetGamepadInstancePlayerIndex(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the player index of a gamepad, or -1 if it's not available

Remarks

This can be called before any gamepads are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadPlayerIndex
- SDL_GetGamepads

SDL_GetGamepadInstanceProduct

Get the USB product ID of a gamepad, if available.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Uint16 SDL_GetGamepadInstanceProduct(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the USB product ID of the selected gamepad. If called on an invalid index, this function returns zero

Remarks

This can be called before any gamepads are opened. If the product ID isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadProduct
- SDL_GetGamepads

SDL_GetGamepadInstanceProductVersion

Get the product version of a gamepad, if available.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Uint16 SDL_GetGamepadInstanceProductVersion(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the product version of the selected gamepad. If called on an invalid index, this function returns zero

Remarks

This can be called before any gamepads are opened. If the product version isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadProductVersion
- SDL_GetGamepads

SDL_GetGamepadInstanceType

Get the type of a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_GamepadType SDL_GetGamepadInstanceType(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the gamepad type.

Remarks

This can be called before any gamepads are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadType
- SDL_GetGamepads
- SDL_GetRealGamepadInstanceType

SDL_GetGamepadInstanceVendor

Get the USB vendor ID of a gamepad, if available.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Uint16 SDL_GetGamepadInstanceVendor(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the USB vendor ID of the selected gamepad. If called on an invalid index, this function returns zero

Remarks

This can be called before any gamepads are opened. If the vendor ID isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadVendor
- SDL_GetGamepads

SDL_GetGamepadJoystick

Get the underlying joystick from a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_Joystick* SDL_GetGamepadJoystick(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object that you want to get a joystick from

Return Value

Returns an SDL_Joystick object; call SDL_GetError() for more information.

Remarks

This function will give you a SDL_Joystick object, which allows you to use the SDL_Joystick functions with a SDL_Gamepad object. This would be useful for getting a joystick's position at any given time, even if it hasn't moved (moving it would produce an event, which would have the axis' value).

The pointer returned is owned by the SDL_Gamepad. You should not call SDL_CloseJoystick() on it, for example, since doing so will likely cause SDL to crash.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadMapping

Get the current mapping of a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
char * SDL_GetGamepadMapping(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad you want to get the current mapping for

Return Value

Returns a string that has the gamepad's mapping or NULL if no mapping is available; call SDL_GetError() for more information.

Remarks

The returned string must be freed with SDL_free().

Details about mappings are discussed with SDL_AddGamepadMapping().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AddGamepadMapping
- SDL_GetGamepadInstanceMapping
- SDL_GetGamepadMappingForGUID
- SDL_SetGamepadMapping

SDL_GetGamepadMappingForGUID

Get the gamepad mapping string for a given GUID.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
char * SDL_GetGamepadMappingForGUID(SDL_JoystickGUID guid);
```

Function Parameters

guid	a structure containing the GUID for which a mapping is desired

Return Value

Returns a mapping string or NULL on error; call SDL_GetError() for more information.

Remarks

The returned string must be freed with SDL_free().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstanceGUID
- SDL_GetJoystickGUID

SDL_GetGamepadMappings

Get the current gamepad mappings.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
char ** SDL_GetGamepadMappings(int *count);
```

Function Parameters

count	a pointer filled in with the number of mappings returned, can be NULL.

Return Value

Returns an array of the mapping strings, NULL-terminated. Must be freed with SDL_free(). Returns NULL on error.

Remarks

You must free the returned pointer with SDL_free() when you are done with it, but you do _not_ free each string in the array.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadName

Get the implementation-dependent name for an opened gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char* SDL_GetGamepadName(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	a gamepad identifier previously returned by SDL_OpenGamepad()

Return Value

Returns the implementation dependent name for the gamepad, or NULL if there is no name or the identifier passed is invalid.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadInstanceName

SDL_GetGamepadPath

Get the implementation-dependent path for an opened gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char* SDL_GetGamepadPath(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	a gamepad identifier previously returned by SDL_OpenGamepad()

Return Value

Returns the implementation dependent path for the gamepad, or NULL if there is no path or the identifier passed is invalid.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadInstancePath

SDL_GetGamepadPlayerIndex

Get the player index of an opened gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_GetGamepadPlayerIndex(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the player index for gamepad, or -1 if it's not available.

Remarks

For XInput gamepads this returns the XInput user index.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetGamepadPlayerIndex

SDL_GetGamepadPowerInfo

Get the battery state of a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_PowerState SDL_GetGamepadPowerInfo(SDL_Gamepad *gamepad, int
    *percent);
```

Function Parameters

gamepad	the gamepad object to query.
percent	a pointer filled in with the percentage of battery life left, between 0 and 100, or NULL to ignore. This will be filled in with -1 we can't determine a value or there is no battery.

Return Value

Returns the current battery state.

Remarks

You should never take a battery status as absolute truth. Batteries (especially failing batteries) are delicate hardware, and the values reported here are best estimates based on what that hardware reports. It's not uncommon for older batteries to lose stored power much faster than it reports, or completely drain when reporting it has 20 percent left, etc.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadProduct

Get the USB product ID of an opened gamepad, if available.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Uint16 SDL_GetGamepadProduct(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the USB product ID, or zero if unavailable.

Remarks

If the product ID isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadInstanceProduct

SDL_GetGamepadProductVersion

Get the product version of an opened gamepad, if available.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Uint16 SDL_GetGamepadProductVersion(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the USB product version, or zero if unavailable.

Remarks

If the product version isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadInstanceProductVersion

SDL_GetGamepadProperties

Get the properties associated with an opened gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_PropertiesID SDL_GetGamepadProperties(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	a gamepad identifier previously returned by SDL_OpenGamepad()

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Remarks

These properties are shared with the underlying joystick object.

The following read-only properties are provided by SDL:

- `SDL_PROP_GAMEPAD_CAP_MONO_LED_BOOLEAN`: true if this gamepad has an LED that has adjustable brightness
- `SDL_PROP_GAMEPAD_CAP_RGB_LED_BOOLEAN`: true if this gamepad has an LED that has adjustable color
- `SDL_PROP_GAMEPAD_CAP_PLAYER_LED_BOOLEAN`: true if this gamepad has a player LED
- `SDL_PROP_GAMEPAD_CAP_RUMBLE_BOOLEAN`: true if this gamepad has left-/right rumble
- `SDL_PROP_GAMEPAD_CAP_TRIGGER_RUMBLE_BOOLEAN`: true if this gamepad has simple trigger rumble

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetGamepads

Get a list of currently connected gamepads.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_JoystickID* SDL_GetGamepads(int *count);
```

Function Parameters

count	a pointer filled in with the number of gamepads returned

Return Value

Returns a 0 terminated array of joystick instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasGamepad
- SDL_OpenGamepad

SDL_GetGamepadSensorData

Get the current state of a gamepad sensor.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_GetGamepadSensorData(SDL_Gamepad *gamepad, SDL_SensorType type,
    float *data, int num_values);
```

Function Parameters

gamepad	The gamepad to query
type	The type of sensor to query
data	A pointer filled with the current sensor state
num_values	The number of values to write to data

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The number of values and interpretation of the data is sensor dependent. See SDL_sensor.h for the details for each type of sensor.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadSensorDataRate

Get the data rate (number of events per second) of a gamepad sensor.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
float SDL_GetGamepadSensorDataRate(SDL_Gamepad *gamepad, SDL_SensorType
    type);
```

Function Parameters

gamepad	The gamepad to query
type	The type of sensor to query

Return Value

Returns the data rate, or 0.0f if the data rate is not available.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadSerial

Get the serial number of an opened gamepad, if available.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char * SDL_GetGamepadSerial(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the serial number, or NULL if unavailable.

Remarks

Returns the serial number of the gamepad, or NULL if it is not available.

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadSteamHandle

Get the Steam Input handle of an opened gamepad, if available.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Uint64 SDL_GetGamepadSteamHandle(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the gamepad handle, or 0 if unavailable.

Remarks

Returns an InputHandle_t for the gamepad that can be used with Steam Input API: https://partner.steamgames.com/doc/api/ISteamInput

Version

This function is available since SDL 3.0.0.

SDL_GetGamepadStringForAxis

Convert from an SDL_GamepadAxis enum to a string.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char * SDL_GetGamepadStringForAxis(SDL_GamepadAxis axis);
```

Function Parameters

axis	an enum value for a given SDL_GamepadAxis

Return Value

Returns a string for the given axis, or NULL if an invalid axis is specified. The string returned is of the format used by SDL_Gamepad mapping strings.

Remarks

The caller should not SDL_free() the returned string.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadAxisFromString

SDL_GetGamepadStringForButton

Convert from an SDL_GamepadButton enum to a string.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char* SDL_GetGamepadStringForButton(SDL_GamepadButton button);
```

Function Parameters

button	an enum value for a given SDL_GamepadButton

Return Value

Returns a string for the given button, or NULL if an invalid button is specified. The string returned is of the format used by SDL_Gamepad mapping strings.

Remarks

The caller should not SDL_free() the returned string.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadButtonFromString

SDL_GetGamepadStringForType

Convert from an SDL_GamepadType enum to a string.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
const char* SDL_GetGamepadStringForType(SDL_GamepadType type);
```

Function Parameters

type	an enum value for a given SDL_GamepadType

Return Value

Returns a string for the given type, or NULL if an invalid type is specified. The string returned is of the format used by SDL_Gamepad mapping strings.

Remarks

The caller should not SDL_free() the returned string.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadTypeFromString

SDL_GetGamepadTouchpadFinger

Get the current state of a finger on a touchpad on a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_GetGamepadTouchpadFinger(SDL_Gamepad *gamepad, int touchpad, int
    finger, Uint8 *state, float *x, float *y, float *pressure);
```

Function Parameters

gamepad	a gamepad
touchpad	a touchpad
finger	a finger
state	filled with state
x	filled with x position
y	filled with y position
pressure	filled with pressure value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumGamepadTouchpadFingers

SDL_GetGamepadType

Get the type of an opened gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_GamepadType SDL_GetGamepadType(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the gamepad type, or SDL_GAMEPAD_TYPE_UNKNOWN if it's not available.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadInstanceType

SDL_GetGamepadTypeFromString

Convert a string into SDL_GamepadType enum.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_GamepadType SDL_GetGamepadTypeFromString(const char *str);
```

Function Parameters

str	string representing a SDL_GamepadType type

Return Value

Returns the SDL_GamepadType enum corresponding to the input string, or `SDL_GAMEPAD_TYPE_UNKNOWN` if no match was found.

Remarks

This function is called internally to translate SDL_Gamepad mapping strings for the underlying joystick device into the consistent SDL_Gamepad mapping. You do not normally need to call this function unless you are parsing SDL_Gamepad mappings in your own code.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadStringForType

SDL_GetGamepadVendor

Get the USB vendor ID of an opened gamepad, if available.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
Uint16 SDL_GetGamepadVendor(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the USB vendor ID, or zero if unavailable.

Remarks

If the vendor ID isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadInstanceVendor

SDL_GetGlobalMouseState

Get the current state of the mouse in relation to the desktop.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
Uint32 SDL_GetGlobalMouseState(float *x, float *y);
```

Function Parameters

x	filled in with the current X coord relative to the desktop; can be NULL
y	filled in with the current Y coord relative to the desktop; can be NULL

Return Value

Returns the current button state as a bitmask which can be tested using the SDL_BUTTON(X) macros.

Remarks

This works similarly to SDL_GetMouseState(), but the coordinates will be reported relative to the top-left of the desktop. This can be useful if you need to track the mouse outside of a specific window and SDL_CaptureMouse() doesn't fit your needs. For example, it could be useful if you need to track the mouse while dragging a window, where coordinates relative to a window might not be in sync at all times.

Note: SDL_GetMouseState() returns the mouse position as SDL understands it from the last pump of the event queue. This function, however, queries the OS for the current mouse position, and as such, might be a slightly less efficient function. Unless you know what you're doing and have a good reason to use this function, you probably want SDL_GetMouseState() instead.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CaptureMouse
- SDL_GetMouseState

SDL_GetGlobalProperties

Get the global SDL properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
SDL_PropertiesID SDL_GetGlobalProperties(void);
```

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetGrabbedWindow

Get the window that currently has an input grab enabled.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_Window* SDL_GetGrabbedWindow(void);
```

Return Value

Returns the window if input is grabbed or NULL otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetWindowMouseGrab
- SDL_SetWindowKeyboardGrab

SDL_GetHapticEffectStatus

Get the status of the current effect on the specified haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_GetHapticEffectStatus(SDL_Haptic *haptic, int effect);
```

Function Parameters

haptic	the SDL_Haptic device to query for the effect status on
effect	the ID of the haptic effect to query its status

Return Value

Returns 0 if it isn't playing, 1 if it is playing, or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Device must support the SDL_HAPTIC_STATUS feature.

Version

This function is available since SDL 3.0.0.

SDL_GetHapticFeatures

Get the haptic device's supported features in bitwise manner.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
Uint32 SDL_GetHapticFeatures(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic device to query

Return Value

Returns a list of supported haptic features in bitwise manner (OR'd), or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HapticEffectSupported
- SDL_GetMaxHapticEffects

SDL_GetHapticFromInstanceID

Get the SDL_Haptic associated with an instance ID, if it has been opened.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_Haptic* SDL_GetHapticFromInstanceID(SDL_HapticID instance_id);
```

Function Parameters

instance_id	the instance ID to get the SDL_Haptic for

Return Value

Returns an SDL_Haptic on success or NULL on failure or if it hasn't been opened yet; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetHapticInstanceID

Get the instance ID of an opened haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_HapticID SDL_GetHapticInstanceID(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic device to query

Return Value

Returns the instance ID of the specified haptic device on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetHapticInstanceName

Get the implementation dependent name of a haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
const char* SDL_GetHapticInstanceName(SDL_HapticID instance_id);
```

Function Parameters

instance_id	the haptic device instance ID

Return Value

Returns the name of the selected haptic device. If no name can be found, this function returns NULL; call SDL_GetError() for more information.

Remarks

This can be called before any haptic devices are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetHapticName
- SDL_OpenHaptic

SDL_GetHapticName

Get the implementation dependent name of a haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
const char* SDL_GetHapticName(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic obtained from SDL_OpenJoystick()

Return Value

Returns the name of the selected haptic device. If no name can be found, this function returns NULL; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetHapticInstanceName

SDL_GetHaptics

Get a list of currently connected haptic devices.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_HapticID* SDL_GetHaptics(int *count);
```

Function Parameters

count	a pointer filled in with the number of haptic devices returned

Return Value

Returns a 0 terminated array of haptic device instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenHaptic

SDL_GetHint

Get the value of a hint.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
const char * SDL_GetHint(const char *name);
```

Function Parameters

name	the hint to query

Return Value

Returns the string value of a hint or NULL if the hint isn't set.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetHint
- SDL_SetHintWithPriority

SDL_GetHintBoolean

Get the boolean value of a hint variable.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
SDL_bool SDL_GetHintBoolean(const char *name, SDL_bool default_value);
```

Function Parameters

name	the name of the hint to get the boolean value from
default_value	the value to return if the hint does not exist

Return Value

Returns the boolean value of a hint or the provided default value if the hint does not exist.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetHint
- SDL_SetHint

SDL_GetIOProperties

Get the properties associated with an SDL_IOStream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_PropertiesID SDL_GetIOProperties(SDL_IOStream *context);
```

Function Parameters

context	a pointer to an SDL_IOStream structure

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetIOSize

Use this function to get the size of the data stream in an SDL_IOStream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
Sint64 SDL_GetIOSize(SDL_IOStream *context);
```

Function Parameters

context	the SDL_IOStream to get the size of the data stream from

Return Value

Returns the size of the data stream in the SDL_IOStream on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetIOStatus

Query the stream status of an SDL_IOStream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_IOStatus SDL_GetIOStatus(SDL_IOStream *context);
```

Function Parameters

context	the SDL_IOStream to query.

Return Value

Returns an SDL_IOStatus enum with the current state.

Remarks

This information can be useful to decide if a short read or write was due to an error, an EOF, or a non-blocking operation that isn't yet ready to complete.

An SDL_IOStream's status is only expected to change after a SDL_ReadIO or SDL_WriteIO call; don't expect it to change if you just call this query function in a tight loop.

Thread Safety

This function should not be called at the same time that another thread is operating on the same SDL_IOStream.

Version

This function is available since SDL 3.0.0.

SDL_GetJoystickAxis

Get the current state of an axis control on a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Sint16 SDL_GetJoystickAxis(SDL_Joystick *joystick, int axis);
```

Function Parameters

joystick	an SDL_Joystick structure containing joystick information
axis	the axis to query; the axis indices start at index 0

Return Value

Returns a 16-bit signed integer representing the current position of the axis or 0 on failure; call SDL_GetError() for more information.

Remarks

SDL makes no promises about what part of the joystick any given axis refers to. Your game should have some sort of configuration UI to let users specify what each axis should be bound to. Alternately, SDL's higher-level Game Controller API makes a great effort to apply order to this lower-level interface, so you know that a specific axis is the "left thumb stick," etc.

The value returned by SDL_GetJoystickAxis() is a signed integer (-32768 to 32767) representing the current position of the axis. It may be necessary to impose certain tolerances on these values to account for jitter.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumJoystickAxes

SDL_GetJoystickAxisInitialState

Get the initial state of an axis control on a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_bool SDL_GetJoystickAxisInitialState(SDL_Joystick *joystick, int
    axis, Sint16 *state);
```

Function Parameters

joystick	an SDL_Joystick structure containing joystick information
axis	the axis to query; the axis indices start at index 0
state	Upon return, the initial value is supplied here.

Return Value

Returns SDL_TRUE if this axis has any initial value, or SDL_FALSE if not.

Remarks

The state is a value ranging from -32768 to 32767.

The axis indices start at index 0.

Version

This function is available since SDL 3.0.0.

SDL_GetJoystickBall

Get the ball axis change since the last poll.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_GetJoystickBall(SDL_Joystick *joystick, int ball, int *dx, int
    *dy);
```

Function Parameters

joystick	the SDL_Joystick to query
ball	the ball index to query; ball indices start at index 0
dx	stores the difference in the x axis position since the last poll
dy	stores the difference in the y axis position since the last poll

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Trackballs can only return relative motion since the last call to SDL_GetJoystickBall(), these motion deltas are placed into `dx` and `dy`.

Most joysticks do not have trackballs.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumJoystickBalls

SDL_GetJoystickButton

Get the current state of a button on a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Uint8 SDL_GetJoystickButton(SDL_Joystick *joystick, int button);
```

Function Parameters

joystick	an SDL_Joystick structure containing joystick information
button	the button index to get the state from; indices start at index 0

Return Value

Returns 1 if the specified button is pressed, 0 otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumJoystickButtons

SDL_GetJoystickConnectionState

Get the connection state of a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickConnectionState SDL_GetJoystickConnectionState(SDL_Joystick
    *joystick);
```

Function Parameters

joystick	The joystick to query

Return Value

Returns the connection state on success or `SDL_JOYSTICK_CONNECTION_INVALID` on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetJoystickFirmwareVersion

Get the firmware version of an opened joystick, if available.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Uint16 SDL_GetJoystickFirmwareVersion(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the firmware version of the selected joystick, or 0 if unavailable.

Remarks

If the firmware version isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

SDL_GetJoystickFromInstanceID

Get the SDL_Joystick associated with an instance ID, if it has been opened.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_Joystick* SDL_GetJoystickFromInstanceID(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the instance ID to get the SDL_Joystick for

Return Value

Returns an SDL_Joystick on success or NULL on failure or if it hasn't been opened yet; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetJoystickFromPlayerIndex

Get the SDL_Joystick associated with a player index.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_Joystick* SDL_GetJoystickFromPlayerIndex(int player_index);
```

Function Parameters

player_index	the player index to get the SDL_Joystick for

Return Value

Returns an SDL_Joystick on success or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickPlayerIndex
- SDL_SetJoystickPlayerIndex

SDL_GetJoystickGUID

Get the implementation-dependent GUID for the joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickGUID SDL_GetJoystickGUID(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the GUID of the given joystick. If called on an invalid index, this function returns a zero GUID; call SDL_GetError() for more information.

Remarks

This function requires an open joystick.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstanceGUID
- SDL_GetJoystickGUIDString

SDL_GetJoystickGUIDFromString

Convert a GUID string into a SDL_JoystickGUID structure.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickGUID SDL_GetJoystickGUIDFromString(const char *pchGUID);
```

Function Parameters

pchGUID	string containing an ASCII representation of a GUID

Return Value

Returns a SDL_JoystickGUID structure.

Remarks

Performs no error checking. If this function is given a string containing an invalid GUID, the function will silently succeed, but the GUID generated will not be useful.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickGUIDString

SDL_GetJoystickGUIDInfo

Get the device information encoded in a SDL_JoystickGUID structure.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
void SDL_GetJoystickGUIDInfo(SDL_JoystickGUID guid, Uint16 *vendor,
    Uint16 *product, Uint16 *version, Uint16 *crc16);
```

Function Parameters

guid	the SDL_JoystickGUID you wish to get info about
vendor	A pointer filled in with the device VID, or 0 if not available
product	A pointer filled in with the device PID, or 0 if not available
version	A pointer filled in with the device version, or 0 if not available
crc16	A pointer filled in with a CRC used to distinguish different products with the same VID/PID, or 0 if not available

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstanceGUID

SDL_GetJoystickGUIDString

Get an ASCII string representation for a given SDL_JoystickGUID.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_GetJoystickGUIDString(SDL_JoystickGUID guid, char *pszGUID, int
    cbGUID);
```

Function Parameters

guid	the SDL_JoystickGUID you wish to convert to string
pszGUID	buffer in which to write the ASCII string
cbGUID	the size of pszGUID

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

You should supply at least 33 bytes for pszGUID.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstanceGUID
- SDL_GetJoystickGUID
- SDL_GetJoystickGUIDFromString

SDL_GetJoystickHat

Get the current state of a POV hat on a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Uint8 SDL_GetJoystickHat(SDL_Joystick *joystick, int hat);


#define SDL_HAT_CENTERED 0x00
#define SDL_HAT_UP       0x01
#define SDL_HAT_RIGHT    0x02
#define SDL_HAT_DOWN     0x04
#define SDL_HAT_LEFT     0x08
#define SDL_HAT_RIGHTUP  (SDL_HAT_RIGHT|SDL_HAT_UP)
#define SDL_HAT_RIGHTDOWN (SDL_HAT_RIGHT|SDL_HAT_DOWN)
#define SDL_HAT_LEFTUP   (SDL_HAT_LEFT|SDL_HAT_UP)
#define SDL_HAT_LEFTDOWN (SDL_HAT_LEFT|SDL_HAT_DOWN)
```

Function Parameters

joystick	an SDL_Joystick structure containing joystick information
hat	the hat index to get the state from; indices start at index 0

Return Value

Returns the current hat position.

Remarks

The returned value will be one of the `SDL_HAT_*` values.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumJoystickHats

SDL_GetJoystickInstanceGUID

Get the implementation-dependent GUID of a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickGUID SDL_GetJoystickInstanceGUID(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the GUID of the selected joystick. If called with an invalid instance_id, this function returns a zero GUID.

Remarks

This can be called before any joysticks are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickGUID
- SDL_GetJoystickGUIDString

SDL_GetJoystickInstanceID

Get the instance ID of an opened joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickID SDL_GetJoystickInstanceID(SDL_Joystick *joystick);
```

Function Parameters

joystick	an SDL_Joystick structure containing joystick information

Return Value

Returns the instance ID of the specified joystick on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetJoystickInstanceName

Get the implementation dependent name of a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
const char* SDL_GetJoystickInstanceName(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the name of the selected joystick. If no name can be found, this function returns NULL; call SDL_GetError() for more information.

Remarks

This can be called before any joysticks are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickName
- SDL_GetJoysticks

SDL_GetJoystickInstancePath

Get the implementation dependent path of a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
const char* SDL_GetJoystickInstancePath(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the path of the selected joystick. If no path can be found, this function returns NULL; call SDL_GetError() for more information.

Remarks

This can be called before any joysticks are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickPath
- SDL_GetJoysticks

SDL_GetJoystickInstancePlayerIndex

Get the player index of a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_GetJoystickInstancePlayerIndex(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the player index of a joystick, or -1 if it's not available

Remarks

This can be called before any joysticks are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickPlayerIndex
- SDL_GetJoysticks

SDL_GetJoystickInstanceProduct

Get the USB product ID of a joystick, if available.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Uint16 SDL_GetJoystickInstanceProduct(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the USB product ID of the selected joystick. If called with an invalid instance_id, this function returns 0.

Remarks

This can be called before any joysticks are opened. If the product ID isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickProduct
- SDL_GetJoysticks

SDL_GetJoystickInstanceProductVersion

Get the product version of a joystick, if available.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Uint16 SDL_GetJoystickInstanceProductVersion(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the product version of the selected joystick. If called with an invalid instance_id, this function returns 0.

Remarks

This can be called before any joysticks are opened. If the product version isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickProductVersion
- SDL_GetJoysticks

SDL_GetJoystickInstanceType

Get the type of a joystick, if available.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickType SDL_GetJoystickInstanceType(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the SDL_JoystickType of the selected joystick. If called with an invalid instance_id, this function returns `SDL_JOYSTICK_TYPE_UNKNOWN`.

Remarks

This can be called before any joysticks are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickType
- SDL_GetJoysticks

SDL_GetJoystickInstanceVendor

Get the USB vendor ID of a joystick, if available.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Uint16 SDL_GetJoystickInstanceVendor(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the USB vendor ID of the selected joystick. If called with an invalid instance_id, this function returns 0.

Remarks

This can be called before any joysticks are opened. If the vendor ID isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickVendor
- SDL_GetJoysticks

SDL_GetJoystickName

Get the implementation dependent name of a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
const char* SDL_GetJoystickName(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the name of the selected joystick. If no name can be found, this function returns NULL; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstanceName

SDL_GetJoystickPath

Get the implementation dependent path of a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
const char* SDL_GetJoystickPath(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the path of the selected joystick. If no path can be found, this function returns NULL; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstancePath

SDL_GetJoystickPlayerIndex

Get the player index of an opened joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_GetJoystickPlayerIndex(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the player index, or -1 if it's not available.

Remarks

For XInput controllers this returns the XInput user index. Many joysticks will not be able to supply this information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetJoystickPlayerIndex

SDL_GetJoystickPowerInfo

Get the battery state of a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_PowerState SDL_GetJoystickPowerInfo(SDL_Joystick *joystick, int
    *percent);
```

Function Parameters

joystick	The joystick to query
percent	a pointer filled in with the percentage of battery life left, between 0 and 100, or NULL to ignore. This will be filled in with -1 we can't determine a value or there is no battery.

Return Value

Returns the current battery state or `SDL_POWERSTATE_ERROR` on failure; call SDL_GetError() for more information.

Remarks

You should never take a battery status as absolute truth. Batteries (especially failing batteries) are delicate hardware, and the values reported here are best estimates based on what that hardware reports. It's not uncommon for older batteries to lose stored power much faster than it reports, or completely drain when reporting it has 20 percent left, etc.

Version

This function is available since SDL 3.0.0.

SDL_GetJoystickProduct

Get the USB product ID of an opened joystick, if available.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Uint16 SDL_GetJoystickProduct(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the USB product ID of the selected joystick, or 0 if unavailable.

Remarks

If the product ID isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstanceProduct

SDL_GetJoystickProductVersion

Get the product version of an opened joystick, if available.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Uint16 SDL_GetJoystickProductVersion(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the product version of the selected joystick, or 0 if unavailable.

Remarks

If the product version isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstanceProductVersion

SDL_GetJoystickProperties

Get the properties associated with a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_PropertiesID SDL_GetJoystickProperties(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Remarks

The following read-only properties are provided by SDL:

- `SDL_PROP_JOYSTICK_CAP_MONO_LED_BOOLEAN`: true if this joystick has an LED that has adjustable brightness
- `SDL_PROP_JOYSTICK_CAP_RGB_LED_BOOLEAN`: true if this joystick has an LED that has adjustable color
- `SDL_PROP_JOYSTICK_CAP_PLAYER_LED_BOOLEAN`: true if this joystick has a player LED
- `SDL_PROP_JOYSTICK_CAP_RUMBLE_BOOLEAN`: true if this joystick has left-/right rumble
- `SDL_PROP_JOYSTICK_CAP_TRIGGER_RUMBLE_BOOLEAN`: true if this joystick has simple trigger rumble

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetJoysticks

Get a list of currently connected joysticks.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickID* SDL_GetJoysticks(int *count);
```

Function Parameters

count	a pointer filled in with the number of joysticks returned

Return Value

Returns a 0 terminated array of joystick instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasJoystick
- SDL_OpenJoystick

SDL_GetJoystickSerial

Get the serial number of an opened joystick, if available.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
const char * SDL_GetJoystickSerial(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the serial number of the selected joystick, or NULL if unavailable.

Remarks

Returns the serial number of the joystick, or NULL if it is not available.

Version

This function is available since SDL 3.0.0.

SDL_GetJoystickType

Get the type of an opened joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_JoystickType SDL_GetJoystickType(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the SDL_JoystickType of the selected joystick.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstanceType

SDL_GetJoystickVendor

Get the USB vendor ID of an opened joystick, if available.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
Uint16 SDL_GetJoystickVendor(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()

Return Value

Returns the USB vendor ID of the selected joystick, or 0 if unavailable.

Remarks

If the vendor ID isn't available this function returns 0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickInstanceVendor

SDL_GetKeyboardFocus

Query the window which currently has keyboard focus.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_Window * SDL_GetKeyboardFocus(void);
```

Return Value

Returns the window with keyboard focus.

Version

This function is available since SDL 3.0.0.

SDL_GetKeyboardInstanceName

Get the name of a keyboard.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
const char* SDL_GetKeyboardInstanceName(SDL_KeyboardID instance_id);
```

Function Parameters

instance_id	the keyboard instance ID

Return Value

Returns the name of the selected keyboard, or NULL on failure; call SDL_GetError() for more information.

Remarks

This function returns "" if the keyboard doesn't have a name.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyboards

SDL_GetKeyboards

Get a list of currently connected keyboards.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_KeyboardID* SDL_GetKeyboards(int *count);
```

Function Parameters

count	a pointer filled in with the number of keyboards returned

Return Value

Returns a 0 terminated array of keyboards instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Remarks

Note that this will include any device or virtual driver that includes keyboard functionality, including some mice, KVM switches, motherboard power buttons, etc. You should wait for input from a device before you consider it actively in use.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyboardInstanceName
- SDL_HasKeyboard

SDL_GetKeyboardState

Get a snapshot of the current state of the keyboard.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
const Uint8* SDL_GetKeyboardState(int *numkeys);
```

Function Parameters

numkeys	if non-NULL, receives the length of the returned array

Return Value

Returns a pointer to an array of key states.

Remarks

The pointer returned is a pointer to an internal SDL array. It will be valid for the whole lifetime of the application and should not be freed by the caller.

A array element with a value of 1 means that the key is pressed and a value of 0 means that it is not. Indexes into this array are obtained by using SDL_Scancode values.

Use SDL_PumpEvents() to update the state array.

This function gives you the current state after all events have been processed, so if a key or button has been pressed and released before you process events, then the pressed state will never show up in the SDL_GetKeyboardState() calls.

Note: This function doesn't take into account whether shift has been pressed or not.

Code Examples

```
const Uint8 *state = SDL_GetKeyboardState(NULL);
if (state[SDL_SCANCODE_RETURN]) {
    printf("<RETURN> is pressed.\n");
}
if (state[SDL_SCANCODE_RIGHT] && state[SDL_SCANCODE_UP]) {
    printf("Right and Up Keys Pressed.\n");
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PumpEvents
- SDL_ResetKeyboard

SDL_GetKeyFromName

Get a key code from a human-readable name.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_Keycode SDL_GetKeyFromName(const char *name);
```

Function Parameters

name	the human-readable key name

Return Value

Returns key code, or `SDLK_UNKNOWN` if the name wasn't recognized; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyFromScancode
- SDL_GetKeyName
- SDL_GetScancodeFromName

SDL_GetKeyFromScancode

Get the key code corresponding to the given scancode according to the current keyboard layout.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_Keycode SDL_GetKeyFromScancode(SDL_Scancode scancode);
```

Function Parameters

scancode	the desired SDL_Scancode to query

Return Value

Returns the SDL_Keycode that corresponds to the given SDL_Scancode.

Remarks

See SDL_Keycode for details.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyName
- SDL_GetScancodeFromKey

SDL_GetKeyName

Get a human-readable name for a key.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
const char* SDL_GetKeyName(SDL_Keycode key);
```

Function Parameters

key	the desired SDL_Keycode to query

Return Value

Returns a pointer to a UTF-8 string that stays valid at least until the next call to this function. If you need it around any longer, you must copy it. If the key doesn't have a name, this function returns an empty string (“”).

Remarks

See SDL_Scancode and SDL_Keycode for details.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyFromName
- SDL_GetKeyFromScancode
- SDL_GetScancodeFromKey

SDL_GetLogOutputFunction

Get the current log output function.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_GetLogOutputFunction(SDL_LogOutputFunction *callback, void
    **userdata);
```

Function Parameters

callback	an SDL_LogOutputFunction filled in with the current log callback
userdata	a pointer filled in with the pointer that is passed to `callback`

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetLogOutputFunction

SDL_GetMasksForPixelFormatEnum

Convert one of the enumerated pixel formats to a bpp value and RGBA masks.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
SDL_bool SDL_GetMasksForPixelFormatEnum(SDL_PixelFormatEnum format,
                                        int *bpp,
                                        Uint32 * Rmask,
                                        Uint32 * Gmask,
                                        Uint32 * Bmask,
                                        Uint32 * Amask);
```

Function Parameters

format	one of the SDL_PixelFormatEnum values
bpp	a bits per pixel value; usually 15, 16, or 32
Rmask	a pointer filled in with the red mask for the format
Gmask	a pointer filled in with the green mask for the format
Bmask	a pointer filled in with the blue mask for the format
Amask	a pointer filled in with the alpha mask for the format

Return Value

Returns SDL_TRUE on success or SDL_FALSE if the conversion wasn't possible; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPixelFormatEnumForMasks

SDL_GetMaxHapticEffects

Get the number of effects a haptic device can store.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_GetMaxHapticEffects(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic device to query

Return Value

Returns the number of effects the haptic device can store or a negative error code on failure; call SDL_GetError() for more information.

Remarks

On some platforms this isn't fully supported, and therefore is an approximation. Always check to see if your created effect was actually created and do not rely solely on SDL_GetMaxHapticEffects().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetMaxHapticEffectsPlaying
- SDL_GetHapticFeatures

SDL_GetMaxHapticEffectsPlaying

Get the number of effects a haptic device can play at the same time.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_GetMaxHapticEffectsPlaying(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic device to query maximum playing effects

Return Value

Returns the number of effects the haptic device can play at the same time or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This is not supported on all platforms, but will always return a value.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetMaxHapticEffects
- SDL_GetHapticFeatures

SDL_GetMemoryFunctions

Get the current set of SDL memory functions.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
void SDL_GetMemoryFunctions(SDL_malloc_func *malloc_func,
                            SDL_calloc_func *calloc_func,
                            SDL_realloc_func *realloc_func,
                            SDL_free_func *free_func);
```

Function Parameters

malloc_func	filled with malloc function
calloc_func	filled with calloc function
realloc_func	filled with realloc function
free_func	filled with free function

Version

This function is available since SDL 3.0.0.

SDL_GetMice

Get a list of currently connected mice.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_MouseID* SDL_GetMice(int *count);
```

Function Parameters

count	a pointer filled in with the number of mice returned

Return Value

Returns a 0 terminated array of mouse instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Remarks

Note that this will include any device or virtual driver that includes mouse functionality, including some game controllers, KVM switches, etc. You should wait for input from a device before you consider it actively in use.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetMouseInstanceName
- SDL_HasMouse

SDL_GetModState

Get the current key modifier state for the keyboard.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_Keymod SDL_GetModState(void);
```

Return Value

Returns an OR'd combination of the modifier keys for the keyboard. See SDL_Keymod for details.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyboardState
- SDL_SetModState

SDL_GetMouseFocus

Get the window which currently has mouse focus.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_Window * SDL_GetMouseFocus(void);
```

Return Value

Returns the window with mouse focus.

Version

This function is available since SDL 3.0.0.

SDL_GetMouseInstanceName

Get the name of a mouse.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
const char* SDL_GetMouseInstanceName(SDL_MouseID instance_id);
```

Function Parameters

instance_id	the mouse instance ID

Return Value

Returns the name of the selected mouse, or NULL on failure; call SDL_GetError() for more information.

Remarks

This function returns "" if the mouse doesn't have a name.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetMice

SDL_GetMouseState

Retrieve the current state of the mouse.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
Uint32 SDL_GetMouseState(float *x, float *y);
```

Function Parameters

x	the x coordinate of the mouse cursor position relative to the focus window
y	the y coordinate of the mouse cursor position relative to the focus window

Return Value

Returns a 32-bit button bitmask of the current button state.

Remarks

The current button state is returned as a button bitmask, which can be tested using the SDL_BUTTON(X) macro (where `X` is generally 1 for the left, 2 for middle, 3 for the right button), and `x` and `y` are set to the mouse cursor position relative to the focus window. You can pass NULL for either `x` or `y`.

Code Examples

```
float x, y;
Uint32 buttons;

SDL_PumpEvents(); // make sure we have the latest mouse state.

buttons = SDL_GetMouseState(&x, &y);

SDL_Log("Mouse cursor is at %f, %f", x, y);
if ((buttons & SDL_BUTTON_LMASK) != 0) {
    SDL_Log("Mouse Button 1 (left) is pressed.");
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGlobalMouseState
- SDL_GetRelativeMouseState

SDL_GetNaturalDisplayOrientation

Get the orientation of a display when it is unrotated.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_DisplayOrientation SDL_GetNaturalDisplayOrientation(SDL_DisplayID
    displayID);
```

Function Parameters

displayID	the instance ID of the display to query

Return Value

Returns The SDL_DisplayOrientation enum value of the display, or `SDL_ORIENTATION_UNKNOWN` if it isn't available.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplays

SDL_GetNumAllocations

Get the number of outstanding (unfreed) allocations.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_GetNumAllocations(void);
```

Return Value

Returns the number of allocations

Version

This function is available since SDL 3.0.0.

SDL_GetNumAudioDrivers

Use this function to get the number of built-in audio drivers.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_GetNumAudioDrivers(void);
```

Return Value

Returns the number of built-in audio drivers.

Remarks

This function returns a hardcoded number. This never returns a negative value; if there are no drivers compiled into this build of SDL, this function returns zero. The presence of a driver in this list does not mean it will function, it just means SDL is capable of interacting with that interface. For example, a build of SDL might have esound support, but if there's no esound server available, SDL's esound driver would fail if used.

By default, SDL tries all drivers, in its preferred order, until one is found to be usable.

Code Examples

```
int i;

for (i = 0; i < SDL_GetNumAudioDrivers(); ++i) {
    printf("Audio driver %d: %s\n", i, SDL_GetAudioDriver(i));
}
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAudioDriver

SDL_GetNumberProperty

Get a number property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
Sint64 SDL_GetNumberProperty(SDL_PropertiesID props, const char *name,
    Sint64 default_value);
```

Function Parameters

props	the properties to query
name	the name of the property to query
default_value	the default value of the property

Return Value

Returns the value of the property, or `default_value` if it is not set or not a number property.

Remarks

You can use SDL_GetPropertyType() to query whether the property exists and is a number property.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPropertyType
- SDL_HasProperty
- SDL_SetNumberProperty

SDL_GetNumCameraDrivers

Use this function to get the number of built-in camera drivers.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
int SDL_GetNumCameraDrivers(void);
```

Return Value

Returns the number of built-in camera drivers.

Remarks

This function returns a hardcoded number. This never returns a negative value; if there are no drivers compiled into this build of SDL, this function returns zero. The presence of a driver in this list does not mean it will function, it just means SDL is capable of interacting with that interface. For example, a build of SDL might have v4l2 support, but if there's no kernel support available, SDL's v4l2 driver would fail if used.

By default, SDL tries all drivers, in its preferred order, until one is found to be usable.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetCameraDriver

SDL_GetNumGamepadTouchpadFingers

Get the number of supported simultaneous fingers on a touchpad on a game gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_GetNumGamepadTouchpadFingers(SDL_Gamepad *gamepad, int touchpad);
```

Function Parameters

gamepad	a gamepad
touchpad	a touchpad

Return Value

Returns number of supported simultaneous fingers

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadTouchpadFinger
- SDL_GetNumGamepadTouchpads

SDL_GetNumGamepadTouchpads

Get the number of touchpads on a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_GetNumGamepadTouchpads(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	a gamepad

Return Value

Returns number of touchpads

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumGamepadTouchpadFingers

SDL_GetNumHapticAxes

Get the number of haptic axes the device has.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_GetNumHapticAxes(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic device to query

Return Value

Returns the number of axes on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The number of haptic axes might be useful if working with the SDL_HapticDirection effect.

Version

This function is available since SDL 3.0.0.

SDL_GetNumJoystickAxes

Get the number of general axis controls on a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_GetNumJoystickAxes(SDL_Joystick *joystick);
```

Function Parameters

joystick	an SDL_Joystick structure containing joystick information

Return Value

Returns the number of axis controls/number of axes on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Often, the directional pad on a game controller will either look like 4 separate buttons or a POV hat, and not axes, but all of this is up to the device and platform.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickAxis
- SDL_GetNumJoystickBalls
- SDL_GetNumJoystickButtons
- SDL_GetNumJoystickHats

SDL_GetNumJoystickBalls

Get the number of trackballs on a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_GetNumJoystickBalls(SDL_Joystick *joystick);
```

Function Parameters

joystick	an SDL_Joystick structure containing joystick information

Return Value

Returns the number of trackballs on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Joystick trackballs have only relative motion events associated with them and their state cannot be polled.

Most joysticks do not have trackballs.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickBall
- SDL_GetNumJoystickAxes
- SDL_GetNumJoystickButtons
- SDL_GetNumJoystickHats

SDL_GetNumJoystickButtons

Get the number of buttons on a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_GetNumJoystickButtons(SDL_Joystick *joystick);
```

Function Parameters

joystick	an SDL_Joystick structure containing joystick information

Return Value

Returns the number of buttons on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickButton
- SDL_GetNumJoystickAxes
- SDL_GetNumJoystickBalls
- SDL_GetNumJoystickHats

SDL_GetNumJoystickHats

Get the number of POV hats on a joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_GetNumJoystickHats(SDL_Joystick *joystick);
```

Function Parameters

joystick	an SDL_Joystick structure containing joystick information

Return Value

Returns the number of POV hats on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickHat
- SDL_GetNumJoystickAxes
- SDL_GetNumJoystickBalls
- SDL_GetNumJoystickButtons

SDL_GetNumRenderDrivers

Get the number of 2D rendering drivers available for the current display.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetNumRenderDrivers(void);
```

Return Value

Returns a number ¿= 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

A render driver is a set of code that handles rendering and texture management on a particular display. Normally there is only one, but some drivers may have several available with different capabilities.

There may be none if SDL was compiled without render support.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateRenderer
- SDL_GetRenderDriver

SDL_GetNumVideoDrivers

Get the number of video drivers compiled into SDL.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetNumVideoDrivers(void);
```

Return Value

Returns a number ¿= 1 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetVideoDriver

SDL_GetOriginalMemoryFunctions

Get the original set of SDL memory functions.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
void SDL_GetOriginalMemoryFunctions(SDL_malloc_func *malloc_func,
                                    SDL_calloc_func *calloc_func,
                                    SDL_realloc_func *realloc_func,
                                    SDL_free_func *free_func);
```

Function Parameters

malloc_func	filled with malloc function
calloc_func	filled with calloc function
realloc_func	filled with realloc function
free_func	filled with free function

Version

This function is available since SDL 3.0.0.

SDL_GetPathInfo

Get information about a filesystem path.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
int SDL_GetPathInfo(const char *path, SDL_PathInfo *info);
```

```
#define SDL_GLOB_CASEINSENSITIVE (1 << 0)
```

Function Parameters

path	the path to query
info	a pointer filled in with information about the path, or NULL to check for the existence of a file

Return Value

Returns 0 on success or a negative error code if the file doesn't exist, or another failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetPenCapabilities

Retrieves capability flags for a given SDL_PenID.

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
Uint32 SDL_GetPenCapabilities(SDL_PenID instance_id,
    SDL_PenCapabilityInfo *capabilities);
```

Function Parameters

instance_id	The pen to query.
capabilities	Detail information about pen capabilities, such as the number of buttons

Return Value

Returns a set of capability flags, cf. SDL_PEN_CAPABILITIES

Version

This function is available since SDL 3.0.0.

SDL_GetPenFromGUID

Retrieves an SDL_PenID for the given SDL_GUID.

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
SDL_PenID SDL_GetPenFromGUID(SDL_GUID guid);
```

Function Parameters

guid	A pen GUID.

Return Value

Returns A valid SDL_PenID, or SDL_PEN_INVALID if there is no matching SDL_PenID.

Version

This function is available since SDL 3.0.0.

SDL_GetPenGUID

Retrieves the SDL_GUID for a given SDL_PenID.

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
SDL_GUID SDL_GetPenGUID(SDL_PenID instance_id);
```

Function Parameters

instance_id	The pen to query.

Return Value

Returns The corresponding pen GUID; persistent across multiple sessions. If "instance_id" is SDL_PEN_INVALID, returns an all-zeroes GUID.

Version

This function is available since SDL 3.0.0.

SDL_GetPenName

Retrieves a human-readable description for a SDL_PenID.

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
const char* SDL_GetPenName(SDL_PenID instance_id);
```

Function Parameters

instance_id	The pen to query.

Return Value

Returns A string that contains the name of the pen, intended for human consumption. The string might or might not be localised, depending on platform settings. It is not guaranteed to be unique; use SDL_GetPenGUID() for (best-effort) unique identifiers. The pointer is managed by the SDL pen subsystem and must not be deallocated. The pointer remains valid until SDL is shut down. Returns NULL on error (cf. SDL_GetError())

Version

This function is available since SDL 3.0.0.

SDL_GetPens

Retrieves all pens that are connected to the system.

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
SDL_PenID* SDL_GetPens(int *count);
```

Function Parameters

count	The number of pens in the array (number of array elements minus 1, i.e., not counting the terminator 0).

Return Value

Returns A 0 terminated array of SDL_PenID values, or NULL on error. The array must be freed with SDL_free(). On a NULL return, SDL_GetError() is set.

Remarks

Yields an array of SDL_PenID values. These identify and track pens throughout a session. To track pens across sessions (program restart), use SDL_GUID .

Version

This function is available since SDL 3.0.0.

SDL_GetPenStatus

Retrieves the pen's current status.

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
Uint32 SDL_GetPenStatus(SDL_PenID instance_id, float *x, float *y, float
    *axes, size_t num_axes);
```

Function Parameters

instance_id	The pen to query.
x	Out-mode parameter for pen x coordinate. May be NULL.
y	Out-mode parameter for pen y coordinate. May be NULL.
axes	Out-mode parameter for axis information. May be null. The axes are in the same order as SDL_PenAxis.
num_axes	Maximum number of axes to write to "axes".

Return Value

Returns a bit mask with the current pen button states (SDL_BUTTON_LMASK etc.), possibly SDL_PEN_DOWN_MASK, and exactly one of SDL_PEN_INK_MASK or SDL_PEN_ERASER_MASK , or 0 on error (see SDL_GetError()).

Remarks

If the pen is detached (cf. SDL_PenConnected), this operation may return default values.

Version

This function is available since SDL 3.0.0.

SDL_GetPenType

Retrieves the pen type for a given SDL_PenID.

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
SDL_PenSubtype SDL_GetPenType(SDL_PenID instance_id);
```

Function Parameters

instance_id	The pen to query.

Return Value

Returns The corresponding pen type (cf. SDL_PenSubtype) or 0 on error. Note that the pen type does not dictate whether the pen tip is SDL_PEN_TIP_INK or SDL_PEN_TIP_ERASER; to determine whether a pen is being used for drawing or in eraser mode, check either the pen tip on SDL_EVENT_PEN_DOWN, or the flag SDL_PEN_ERASER_MASK in the pen state.

Version

This function is available since SDL 3.0.0.

SDL_GetPerformanceCounter

Get the current value of the high resolution counter.

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
Uint64 SDL_GetPerformanceCounter(void);
```

Return Value

Returns the current counter value.

Remarks

This function is typically used for profiling.

The counter values are only meaningful relative to each other. Differences between values can be converted to times by using SDL_GetPerformanceFrequency().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPerformanceFrequency

SDL_GetPerformanceFrequency

Get the count per second of the high resolution counter.

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
Uint64 SDL_GetPerformanceFrequency(void);
```

Return Value

Returns a platform-specific count per second.

Code Examples

```
#include <SDL3/SDL.h>

#define DEFAULT_RESOLUTION 1

static int ticks = 0;

static Uint32 SDLCALL
ticktock(Uint32 interval, void *param)
{
    ++ticks;
    return (interval);
}

static Uint32 SDLCALL
callback(Uint32 interval, void *param)
{
    SDL_Log("Timer %d : param = %d", interval, (int) (uintptr_t) param);
    return interval;
}

int
main(int argc, char *argv[])
{
    int i, desired;
    SDL_TimerID t1, t2, t3;
    Uint32 start32, now32;
    Uint64 start, now;
```

```
/* Enable standard application logging */
SDL_LogSetPriority(SDL_LOG_CATEGORY_APPLICATION,
    SDL_LOG_PRIORITY_INFO);

if (SDL_Init(SDL_INIT_TIMER) < 0) {
    SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't initialize
        SDL: %s", SDL_GetError());
    return (1);
}

/* Start the timer */
desired = 0;
if (argv[1]) {
    desired = SDL_atoi(argv[1]);
}
if (desired == 0) {
    desired = DEFAULT_RESOLUTION;
}
t1 = SDL_AddTimer(desired, ticktock, NULL);

/* Wait 10 seconds */
SDL_Log("Waiting 10 seconds");
SDL_Delay(10 * 1000);

/* Stop the timer */
SDL_RemoveTimer(t1);

/* Print the results */
if (ticks) {
    SDL_Log("Timer resolution: desired = %d ms, actual = %f ms",
            desired, (double) (10 * 1000) / ticks);
}

/* Test multiple timers */
SDL_Log("Testing multiple timers...");
t1 = SDL_AddTimer(100, callback, (void *) 1);
if (!t1)
    SDL_LogError(SDL_LOG_CATEGORY_APPLICATION,"Could not create
        timer 1: %s", SDL_GetError());
t2 = SDL_AddTimer(50, callback, (void *) 2);
if (!t2)
    SDL_LogError(SDL_LOG_CATEGORY_APPLICATION,"Could not create
        timer 2: %s", SDL_GetError());
t3 = SDL_AddTimer(233, callback, (void *) 3);
if (!t3)
    SDL_LogError(SDL_LOG_CATEGORY_APPLICATION,"Could not create
        timer 3: %s", SDL_GetError());

/* Wait 10 seconds */
SDL_Log("Waiting 10 seconds");
```

```
    SDL_Delay(10 * 1000);

    SDL_Log("Removing timer 1 and waiting 5 more seconds");
    SDL_RemoveTimer(t1);

    SDL_Delay(5 * 1000);

    SDL_RemoveTimer(t2);
    SDL_RemoveTimer(t3);

    start = SDL_GetPerformanceCounter();
    for (i = 0; i < 1000000; ++i) {
        ticktock(0, NULL);
    }
    now = SDL_GetPerformanceCounter();
    SDL_Log("1 million iterations of ticktock took %f ms", (double)((now
        - start)*1000) / SDL_GetPerformanceFrequency());

    SDL_Log("Performance counter frequency: %"SDL_PRIu64"", (unsigned
        long long) SDL_GetPerformanceFrequency());
    start32 = SDL_GetTicks();
    start = SDL_GetPerformanceCounter();
    SDL_Delay(1000);
    now = SDL_GetPerformanceCounter();
    now32 = SDL_GetTicks();
    SDL_Log("Delay 1 second = %d ms in ticks, %f ms according to
        performance counter", (now32-start32), (double)((now -
        start)*1000) / SDL_GetPerformanceFrequency());

    SDL_Quit();
    return (0);
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPerformanceCounter

SDL_GetPixelFormatEnumForMasks

Convert a bpp value and RGBA masks to an enumerated pixel format.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
SDL_PixelFormatEnum SDL_GetPixelFormatEnumForMasks(int bpp,
                                Uint32 Rmask,
                                Uint32 Gmask,
                                Uint32 Bmask,
                                Uint32 Amask);
```

Function Parameters

bpp	a bits per pixel value; usually 15, 16, or 32
Rmask	the red mask for the format
Gmask	the green mask for the format
Bmask	the blue mask for the format
Amask	the alpha mask for the format

Return Value

Returns the SDL_PixelFormatEnum value corresponding to the format masks, or SDL_PIXELFORMAT_UNKNOWN if there isn't a match.

Remarks

This will return `SDL_PIXELFORMAT_UNKNOWN` if the conversion wasn't possible.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetMasksForPixelFormatEnum

SDL_GetPixelFormatName

Get the human readable name of a pixel format.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
const char* SDL_GetPixelFormatName(SDL_PixelFormatEnum format);
```

Function Parameters

format	the pixel format to query

Return Value

Returns the human readable name of the specified pixel format or `SDL_PIXELFORMAT_UNKNOWN` if the format isn't recognized.

Code Examples

```
SDL_Surface* surface; // a valid surface from wherever.
SDL_PixelFormat* pixelFormat = surface->format;
Uint32 pixelFormatEnum = pixelFormat->format;
const char* surfacePixelFormatName =
    SDL_GetPixelFormatName(pixelFormatEnum);
SDL_Log("The surface's pixelformat is %s", surfacePixelFormatName);
// prints something like "The surface's pixelformat is
    SDL_PIXELFORMAT_ABGR8888"
```

Version

This function is available since SDL 3.0.0.

SDL_GetPlatform

Get the name of the platform.

Header File

Defined in SDL3/SDL_platform.h

Syntax

```
const char * SDL_GetPlatform (void);
```

Return Value

Returns the name of the platform. If the correct platform name is not available, returns a string beginning with the text "Unknown".

Remarks

Here are the names returned for some (but not all) supported platforms:

- "Windows"
- "macOS"
- "Linux"
- "iOS"
- "Android"

Version

This function is available since SDL 3.0.0.

SDL_GetPowerInfo

Get the current power supply details.

Header File

Defined in SDL3/SDL_power.h

Syntax

```
SDL_PowerState SDL_GetPowerInfo(int *seconds, int *percent);
```

Function Parameters

seconds	a pointer filled in with the seconds of battery life left, or NULL to ignore. This will be filled in with -1 if we can't determine a value or there is no battery.
percent	a pointer filled in with the percentage of battery life left, between 0 and 100, or NULL to ignore. This will be filled in with -1 we can't determine a value or there is no battery.

Return Value

Returns the current battery state or `SDL_POWERSTATE_ERROR` on failure; call SDL_GetError() for more information.

Remarks

You should never take a battery status as absolute truth. Batteries (especially failing batteries) are delicate hardware, and the values reported here are best estimates based on what that hardware reports. It's not uncommon for older batteries to lose stored power much faster than it reports, or completely drain when reporting it has 20 percent left, etc.

Battery status can change at any time; if you are concerned with power state, you should call this function frequently, and perhaps ignore changes until they seem to be stable for a few seconds.

It's possible a platform can only report battery percentage or time left but not both.

Code Examples

```
int secs, pct;
if (SDL_GetPowerInfo(&secs, &pct) == SDL_POWERSTATE_ON_BATTERY) {
    printf("Battery is draining: ");
    if (secs == -1) {
```

```
        printf("(unknown time left)\n");
    } else {
        printf("(%d seconds left)\n", secs);
    }

    if (pct == -1) {
        printf("(unknown percentage left)\n");
    } else {
        printf("(%d percent left)\n", pct);
    }
}
```

Version

This function is available since SDL 3.0.0.

SDL_GetPreferredLocales

Report the user's preferred locale.

Header File

Defined in SDL3/SDL_locale.h

Syntax

```
SDL_Locale * SDL_GetPreferredLocales(void);
```

Return Value

Returns array of locales, terminated with a locale with a NULL language field. Will return NULL on error; call SDL_GetError() for more information.

Remarks

This returns an array of SDL_Locale structs, the final item zeroed out. When the caller is done with this array, it should call SDL_free() on the returned value; all the memory involved is allocated in a single block, so a single SDL_free() will suffice.

Returned language strings are in the format xx, where 'xx' is an ISO-639 language specifier (such as "en" for English, "de" for German, etc). Country strings are in the format YY, where "YY" is an ISO-3166 country code (such as "US" for the United States, "CA" for Canada, etc). Country might be NULL if there's no specific guidance on them (so you might get "en", "US" for American English, but "en", NULL means "English language, generically"). Language strings are never NULL, except to terminate the array.

Please note that not all of these strings are 2 characters; some are three or more.

The returned list of locales are in the order of the user's preference. For example, a German citizen that is fluent in US English and knows enough Japanese to navigate around Tokyo might have a list like: "de", "en_US", "jp", NULL . Someone from England might prefer British English (where "color" is spelled "colour", etc), but will settle for anything like it: "en_GB", "en", NULL .

This function returns NULL on error, including when the platform does not supply this information at all.

This might be a "slow" call that has to query the operating system. It's best to ask for this once and save the results. However, this list can change, usually because the user has changed a system preference outside of your program; SDL will send an SDL_EVENT_LOCALE_CHANGED event in this case, if possible, and you can call this function again to get an updated copy of preferred locales.

Version

This function is available since SDL 3.0.0.

SDL_GetPrefPath

Get the user-and-app-specific path where files can be written.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
char* SDL_GetPrefPath(const char *org, const char *app);
```

Function Parameters

org	the name of your organization
app	the name of your application

Return Value

Returns a UTF-8 string of the user directory in platform-dependent notation. NULL if there's a problem (creating directory failed, etc.).

Remarks

Get the "pref dir". This is meant to be where users can write personal files (preferences and save games, etc) that are specific to your application. This directory is unique per user, per application.

This function will decide the appropriate location in the native filesystem, create the directory if necessary, and return a string of the absolute path to the directory in UTF-8 encoding.

On Windows, the string might look like: `C:`
`Users`
`bob`
`AppData`
`Roaming`
`My Company`
`My Program Name`

On Linux, the string might look like: `/home/bob/.local/share/My Program Name/`

On macOS, the string might look like: `/Users/bob/Library/Application Support/My Program Name/`

You should assume the path returned by this function is the only safe place to write files (and that SDL_GetBasePath(), while it might be writable, or even the parent of the returned path, isn't where you should be writing things).

Both the org and app strings may become part of a directory name, so please follow these rules:

- Try to use the same org string (_including case-sensitivity_) for all your applications that use this function.
- Always use a unique app string for each one, and make sure it never changes for an app once you've decided on it.
- Unicode characters are legal, as long as it's UTF-8 encoded, but...
- ...only use letters, numbers, and spaces. Avoid punctuation like "Game Name 2: Bad Guy's Revenge!" ... "Game Name 2" is sufficient.

The returned path is guaranteed to end with a path separator ('
' on Windows, '/' on most other platforms).

The pointer returned is owned by the caller. Please call SDL_free() on the pointer when done with it.

Code Examples

```
char *pref_path = NULL;

void InitializePrefPath() {
    char *base_path = SDL_GetPrefPath("My Company", "My Awesome SDL 2
        Game");
    if (base_path) {
        pref_path = base_path;
    } else {
        /* Do something to disable writing in-game */
    }
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetBasePath

SDL_GetPrimaryDisplay

Return the primary display.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_DisplayID SDL_GetPrimaryDisplay(void);
```

Return Value

Returns the instance ID of the primary display on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetDisplays

SDL_GetPrimarySelectionText

Get UTF-8 text from the primary selection, which must be freed with SDL_free().

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
char * SDL_GetPrimarySelectionText(void);
```

Return Value

Returns the primary selection text on success or an empty string on failure; call SDL_GetError() for more information. Caller must call SDL_free() on the returned pointer when done with it (even if there was an error).

Remarks

This functions returns empty string if there was not enough memory left for a copy of the primary selection's content.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasPrimarySelectionText
- SDL_SetPrimarySelectionText

SDL_GetProperty

Get a property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
void* SDL_GetProperty(SDL_PropertiesID props, const char *name, void
    *default_value);
```

Function Parameters

props	the properties to query
name	the name of the property to query
default_value	the default value of the property

Return Value

Returns the value of the property, or `default_value` if it is not set or not a pointer property.

Remarks

By convention, the names of properties that SDL exposes on objects will start with "SDL.", and properties that SDL uses internally will start with "SDL.internal.". These should be considered read-only and should not be modified by applications.

Thread Safety

It is safe to call this function from any thread, although the data returned is not protected and could potentially be freed if you call SDL_SetProperty() or SDL_ClearProperty() on these properties from another thread. If you need to avoid this, use SDL_LockProperties() and SDL_UnlockProperties().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetBooleanProperty
- SDL_GetFloatProperty
- SDL_GetNumberProperty
- SDL_GetPropertyType
- SDL_GetStringProperty
- SDL_HasProperty
- SDL_SetProperty

SDL_GetPropertyType

Get the type of a property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
SDL_PropertyType SDL_GetPropertyType(SDL_PropertiesID props, const char
    *name);
```

Function Parameters

props	the properties to query
name	the name of the property to query

Return Value

Returns the type of the property, or SDL_PROPERTY_TYPE_INVALID if it is not set.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasProperty

SDL_GetRealGamepadInstanceType

Get the type of a gamepad, ignoring any mapping override.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_GamepadType SDL_GetRealGamepadInstanceType(SDL_JoystickID
    instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns the gamepad type.

Remarks

This can be called before any gamepads are opened.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadInstanceType
- SDL_GetGamepads
- SDL_GetRealGamepadType

SDL_GetRealGamepadType

Get the type of an opened gamepad, ignoring any mapping override.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_GamepadType SDL_GetRealGamepadType(SDL_Gamepad *gamepad);
```

Function Parameters

gamepad	the gamepad object to query.

Return Value

Returns the gamepad type, or SDL_GAMEPAD_TYPE_UNKNOWN if it's not available.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRealGamepadInstanceType

SDL_GetRectAndLineIntersection

Calculate the intersection of a rectangle and line segment.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_bool SDL_GetRectAndLineIntersection(const SDL_Rect *
                                        rect, int *X1,
                                        int *Y1, int *X2,
                                        int *Y2);
```

Function Parameters

rect	an SDL_Rect structure representing the rectangle to intersect
X1	a pointer to the starting X-coordinate of the line
Y1	a pointer to the starting Y-coordinate of the line
X2	a pointer to the ending X-coordinate of the line
Y2	a pointer to the ending Y-coordinate of the line

Return Value

Returns SDL_TRUE if there is an intersection, SDL_FALSE otherwise.

Remarks

This function is used to clip a line segment to a rectangle. A line segment contained entirely within the rectangle or that does not intersect will remain unchanged. A line segment that crosses the rectangle at either or both ends will be clipped to the boundary of the rectangle and the new coordinates saved in `X1`, `Y1`, `X2`, and/or `Y2` as necessary.

Version

This function is available since SDL 3.0.0.

SDL_GetRectAndLineIntersectionFloat

Calculate the intersection of a rectangle and line segment with float precision.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_bool SDL_GetRectAndLineIntersectionFloat(const SDL_FRect *
                                rect, float *X1,
                                float *Y1, float *X2,
                                float *Y2);
```

Function Parameters

rect	an SDL_FRect structure representing the rectangle to intersect
X1	a pointer to the starting X-coordinate of the line
Y1	a pointer to the starting Y-coordinate of the line
X2	a pointer to the ending X-coordinate of the line
Y2	a pointer to the ending Y-coordinate of the line

Return Value

Returns SDL_TRUE if there is an intersection, SDL_FALSE otherwise.

Remarks

This function is used to clip a line segment to a rectangle. A line segment contained entirely within the rectangle or that does not intersect will remain unchanged. A line segment that crosses the rectangle at either or both ends will be clipped to the boundary of the rectangle and the new coordinates saved in `X1`, `Y1`, `X2`, and/or `Y2` as necessary.

Version

This function is available since SDL 3.0.0.

SDL_GetRectEnclosingPoints

Calculate a minimal rectangle enclosing a set of points.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_bool SDL_GetRectEnclosingPoints(const SDL_Point * points,
                        int count,
                        const SDL_Rect * clip,
                        SDL_Rect * result);
```

Function Parameters

points	an array of SDL_Point structures representing points to be enclosed
count	the number of structures in the `points` array
clip	an SDL_Rect used for clipping or NULL to enclose all points
result	an SDL_Rect structure filled in with the minimal enclosing rectangle

Return Value

Returns SDL_TRUE if any points were enclosed or SDL_FALSE if all the points were outside of the clipping rectangle.

Remarks

If `clip` is not NULL then only points inside of the clipping rectangle are considered.

Version

This function is available since SDL 3.0.0.

SDL_GetRectEnclosingPointsFloat

Calculate a minimal rectangle enclosing a set of points with float precision.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_bool SDL_GetRectEnclosingPointsFloat(const SDL_FPoint * points,
                         int count,
                         const SDL_FRect * clip,
                         SDL_FRect * result);
```

Function Parameters

points	an array of SDL_FPoint structures representing points to be enclosed
count	the number of structures in the `points` array
clip	an SDL_FRect used for clipping or NULL to enclose all points
result	an SDL_FRect structure filled in with the minimal enclosing rectangle

Return Value

Returns SDL_TRUE if any points were enclosed or SDL_FALSE if all the points were outside of the clipping rectangle.

Remarks

If `clip` is not NULL then only points inside of the clipping rectangle are considered.

Version

This function is available since SDL 3.0.0.

SDL_GetRectIntersection

Calculate the intersection of two rectangles.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_bool SDL_GetRectIntersection(const SDL_Rect * A,
                                 const SDL_Rect * B,
                                 SDL_Rect * result);
```

Function Parameters

A	an SDL_Rect structure representing the first rectangle
B	an SDL_Rect structure representing the second rectangle
result	an SDL_Rect structure filled in with the intersection of rectangles `A` and `B`

Return Value

Returns SDL_TRUE if there is an intersection, SDL_FALSE otherwise.

Remarks

If `result` is NULL then this function will return SDL_FALSE.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasRectIntersection

SDL_GetRectIntersectionFloat

Calculate the intersection of two rectangles with float precision.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_bool SDL_GetRectIntersectionFloat(const SDL_FRect * A,
                          const SDL_FRect * B,
                          SDL_FRect * result);
```

Function Parameters

A	an SDL_FRect structure representing the first rectangle
B	an SDL_FRect structure representing the second rectangle
result	an SDL_FRect structure filled in with the intersection of rectangles `A` and `B`

Return Value

Returns SDL_TRUE if there is an intersection, SDL_FALSE otherwise.

Remarks

If `result` is NULL then this function will return SDL_FALSE.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasRectIntersectionFloat

SDL_GetRectUnion

Calculate the union of two rectangles.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
int SDL_GetRectUnion(const SDL_Rect * A,
                     const SDL_Rect * B,
                     SDL_Rect * result);
```

Function Parameters

A	an SDL_Rect structure representing the first rectangle
B	an SDL_Rect structure representing the second rectangle
result	an SDL_Rect structure filled in with the union of rectangles `A` and `B`

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetRectUnionFloat

Calculate the union of two rectangles with float precision.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
int SDL_GetRectUnionFloat(const SDL_FRect * A,
                  const SDL_FRect * B,
                  SDL_FRect * result);
```

Function Parameters

A	an SDL_FRect structure representing the first rectangle
B	an SDL_FRect structure representing the second rectangle
result	an SDL_FRect structure filled in with the union of rectangles `A` and `B`

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetRelativeMouseMode

Query whether relative mouse mode is enabled.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_bool SDL_GetRelativeMouseMode(void);
```

Return Value

Returns SDL_TRUE if relative mode is enabled or SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRelativeMouseMode

SDL_GetRelativeMouseState

Retrieve the relative state of the mouse.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
Uint32 SDL_GetRelativeMouseState(float *x, float *y);
```

Function Parameters

x	a pointer filled with the last recorded x coordinate of the mouse
y	a pointer filled with the last recorded y coordinate of the mouse

Return Value

Returns a 32-bit button bitmask of the relative button state.

Remarks

The current button state is returned as a button bitmask, which can be tested using the `SDL_BUTTON(X)` macros (where `X` is generally 1 for the left, 2 for middle, 3 for the right button), and `x` and `y` are set to the mouse deltas since the last call to SDL_GetRelativeMouseState() or since event initialization. You can pass NULL for either `x` or `y`.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetMouseState

SDL_GetRenderClipRect

Get the clip rectangle for the current target.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderClipRect(SDL_Renderer *renderer, SDL_Rect *rect);
```

Function Parameters

renderer	the rendering context
rect	an SDL_Rect structure filled in with the current clipping area or an empty rectangle if clipping is disabled

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderClipEnabled
- SDL_SetRenderClipRect

SDL_GetRenderColorScale

Get the color scale used for render operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderColorScale(SDL_Renderer *renderer, float *scale);
```

Function Parameters

renderer	the rendering context
scale	a pointer filled in with the current color scale value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderColorScale

SDL_GetRenderDrawBlendMode

Get the blend mode used for drawing operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderDrawBlendMode(SDL_Renderer *renderer, SDL_BlendMode
    *blendMode);
```

Function Parameters

renderer	the rendering context
blendMode	a pointer filled in with the current SDL_BlendMode

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderDrawBlendMode

SDL_GetRenderDrawColor

Get the color used for drawing operations (Rect, Line and Clear).

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderDrawColor(SDL_Renderer *renderer, Uint8 *r, Uint8 *g,
    Uint8 *b, Uint8 *a);
```

Function Parameters

renderer	the rendering context
r	a pointer filled in with the red value used to draw on the rendering target
g	a pointer filled in with the green value used to draw on the rendering target
b	a pointer filled in with the blue value used to draw on the rendering target
a	a pointer filled in with the alpha value used to draw on the rendering target; usually `SDL_ALPHA_OPAQUE` (255)

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderDrawColorFloat
- SDL_SetRenderDrawColor

SDL_GetRenderDrawColorFloat

Get the color used for drawing operations (Rect, Line and Clear).

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderDrawColorFloat(SDL_Renderer *renderer, float *r, float
    *g, float *b, float *a);
```

Function Parameters

renderer	the rendering context
r	a pointer filled in with the red value used to draw on the rendering target
g	a pointer filled in with the green value used to draw on the rendering target
b	a pointer filled in with the blue value used to draw on the rendering target
a	a pointer filled in with the alpha value used to draw on the rendering target

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderDrawColorFloat
- SDL_GetRenderDrawColor

SDL_GetRenderDriver

Use this function to get the name of a built in 2D rendering driver.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
const char* SDL_GetRenderDriver(int index);
```

Function Parameters

index	the index of the rendering driver; the value ranges from 0 to SDL_GetNumRenderDrivers() - 1

Return Value

Returns the name of the rendering driver at the requested index, or NULL if an invalid index was specified.

Remarks

The list of rendering drivers is given in the order that they are normally initialized by default; the drivers that seem more reasonable to choose first (as far as the SDL developers believe) are earlier in the list.

The names of drivers are all simple, low-ASCII identifiers, like "opengl", "direct3d12" or "metal". These never have Unicode characters, and are not meant to be proper names.

The returned value points to a static, read-only string; do not modify or free it!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumRenderDrivers

SDL_GetRenderer

Get the renderer associated with a window.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Renderer* SDL_GetRenderer(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the rendering context on success or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetRendererFromTexture

Get the renderer that created an SDL_Texture.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Renderer* SDL_GetRendererFromTexture(SDL_Texture *texture);
```

Function Parameters

texture	the texture to query

Return Value

Returns a pointer to the SDL_Renderer that created the texture, or NULL on failure; call SDL_GetError() for more information.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_GetRendererInfo

Get information about a rendering context.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRendererInfo(SDL_Renderer *renderer, SDL_RendererInfo *info);
```

Function Parameters

renderer	the rendering context
info	an SDL_RendererInfo structure filled with information about the current renderer

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateRenderer
- SDL_CreateRendererWithProperties

SDL_GetRendererProperties

Get the properties associated with a renderer.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_PropertiesID SDL_GetRendererProperties(SDL_Renderer *renderer);
```

Function Parameters

renderer	the rendering context

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Remarks

The following read-only properties are provided by SDL:

- `SDL_PROP_RENDERER_NAME_STRING`: the name of the rendering driver
- `SDL_PROP_RENDERER_WINDOW_POINTER`: the window where rendering is displayed, if any
- `SDL_PROP_RENDERER_SURFACE_POINTER`: the surface where rendering is displayed, if this is a software renderer without a window
- `SDL_PROP_RENDERER_OUTPUT_COLORSPACE_NUMBER`: an SDL_ColorSpace value describing the colorspace for output to the display, defaults to SDL_COLORSPACE_SRGB.
- `SDL_PROP_RENDERER_HDR_ENABLED_BOOLEAN`: true if the output colorspace is SDL_COLORSPACE_SRGB_LINEAR and the renderer is showing on a display with HDR enabled. This property can change dynamically when SDL_EVENT_DISPLAY_HDR_STATE_CHANGED is sent.
- `SDL_PROP_RENDERER_SDR_WHITE_POINT_FLOAT`: the value of SDR white in the SDL_COLORSPACE_SRGB_LINEAR colorspace. When HDR is enabled, this value is automatically multiplied into the color scale. This property can change dynamically when SDL_EVENT_DISPLAY_HDR_STATE_CHANGED is sent.

- SDL_PROP_RENDERER_HDR_HEADROOM_FLOAT: the additional high dynamic range that can be displayed, in terms of the SDR white point. When HDR is not enabled, this will be 1.0. This property can change dynamically when SDL_EVENT_DISPLAY_HDR_STATE_CHANGED is sent.

With the direct3d renderer:

- SDL_PROP_RENDERER_D3D9_DEVICE_POINTER: the IDirect3DDevice9 associated with the renderer

With the direct3d11 renderer:

- SDL_PROP_RENDERER_D3D11_DEVICE_POINTER: the ID3D11Device associated with the renderer

With the direct3d12 renderer:

- SDL_PROP_RENDERER_D3D12_DEVICE_POINTER: the ID3D12Device associated with the renderer
- SDL_PROP_RENDERER_D3D12_COMMAND_QUEUE_POINTER: the ID3D12CommandQueue associated with the renderer

With the vulkan renderer:

- SDL_PROP_RENDERER_VULKAN_INSTANCE_POINTER: the VkInstance associated with the renderer
- SDL_PROP_RENDERER_VULKAN_SURFACE_NUMBER: the VkSurfaceKHR associated with the renderer
- SDL_PROP_RENDERER_VULKAN_PHYSICAL_DEVICE_POINTER: the VkPhysicalDevice associated with the renderer
- SDL_PROP_RENDERER_VULKAN_DEVICE_POINTER: the VkDevice associated with the renderer
- SDL_PROP_RENDERER_VULKAN_GRAPHICS_QUEUE_FAMILY_INDEX_NUMBER: the queue family index used for rendering
- SDL_PROP_RENDERER_VULKAN_PRESENT_QUEUE_FAMILY_INDEX_NUMBER: the queue family index used for presentation
- SDL_PROP_RENDERER_VULKAN_SWAPCHAIN_IMAGE_COUNT_NUMBER: the number of swapchain images, or potential frames in flight, used by the Vulkan renderer

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetRenderLogicalPresentation

Get device independent resolution and presentation mode for rendering.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderLogicalPresentation(SDL_Renderer *renderer, int *w, int
    *h, SDL_RendererLogicalPresentation *mode, SDL_ScaleMode
    *scale_mode);
```

Function Parameters

renderer	the rendering context
w	an int to be filled with the width
h	an int to be filled with the height
mode	a pointer filled in with the presentation mode
scale_mode	a pointer filled in with the scale mode

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function gets the width and height of the logical rendering output, or the output size in pixels if a logical resolution is not enabled.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderLogicalPresentation

SDL_GetRenderMetalCommandEncoder

Get the Metal command encoder for the current frame.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
void* SDL_GetRenderMetalCommandEncoder(SDL_Renderer *renderer);
```

Function Parameters

renderer	The renderer to query

Return Value

Returns an `id<MTLRenderCommandEncoder>` on success, or NULL if the renderer isn't a Metal renderer or there was an error.

Remarks

This function returns `void *`, so SDL doesn't have to include Metal's headers, but it can be safely cast to an `id<MTLRenderCommandEncoder>`.

This will return NULL if Metal refuses to give SDL a drawable to render to, which might happen if the window is hidden/minimized/offscreen. This doesn't apply to command encoders for render targets, just the window's backbuffer. Check your return values!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderMetalLayer

SDL_GetRenderMetalLayer

Get the CAMetalLayer associated with the given Metal renderer.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
void* SDL_GetRenderMetalLayer(SDL_Renderer *renderer);
```

Function Parameters

renderer	The renderer to query

Return Value

Returns a `CAMetalLayer *` on success, or NULL if the renderer isn't a Metal renderer

Remarks

This function returns `void , so SDL doesn't have to include Metal's headers, but it can be safely cast to a CAMetalLayer .`

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderMetalCommandEncoder

SDL_GetRenderOutputSize

Get the output size in pixels of a rendering context.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderOutputSize(SDL_Renderer *renderer, int *w, int *h);
```

Function Parameters

renderer	the rendering context
w	a pointer filled in with the width in pixels
h	a pointer filled in with the height in pixels

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This returns the true output size in pixels, ignoring any render targets or logical size and presentation.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetCurrentRenderOutputSize

SDL_GetRenderScale

Get the drawing scale for the current target.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderScale(SDL_Renderer *renderer, float *scaleX, float
    *scaleY);
```

Function Parameters

renderer	the rendering context
scaleX	a pointer filled in with the horizontal scaling factor
scaleY	a pointer filled in with the vertical scaling factor

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderScale

SDL_GetRenderTarget

Get the current render target.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Texture* SDL_GetRenderTarget(SDL_Renderer *renderer);
```

Function Parameters

renderer	the rendering context

Return Value

Returns the current render target or NULL for the default render target.

Remarks

The default render target is the window for which the renderer was created, and is reported a NULL here.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderTarget

SDL_GetRenderViewport

Get the drawing area for the current target.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderViewport(SDL_Renderer *renderer, SDL_Rect *rect);
```

Function Parameters

renderer	the rendering context
rect	an SDL_Rect structure filled in with the current drawing area

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderViewportSet
- SDL_SetRenderViewport

SDL_GetRenderVSync

Get VSync of the given renderer.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetRenderVSync(SDL_Renderer *renderer, int *vsync);
```

Function Parameters

renderer	The renderer to toggle
vsync	an int filled with 1 for on, 0 for off. All other values are reserved

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderVSync

SDL_GetRenderWindow

Get the window associated with a renderer.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Window* SDL_GetRenderWindow(SDL_Renderer *renderer);
```

Function Parameters

renderer	the renderer to query

Return Value

Returns the window on success or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetRevision

Get the code revision of SDL that is linked against your program.

Header File

Defined in SDL3/SDL_version.h

Syntax

```
const char* SDL_GetRevision(void);
```

Return Value

Returns an arbitrary string, uniquely identifying the exact revision of the SDL library in use.

Remarks

This value is the revision of the code you are linked with and may be different from the code you are compiling with, which is found in the constant SDL_REVISION.

The revision is arbitrary string (a hash value) uniquely identifying the exact revision of the SDL library in use, and is only useful in comparing against other revisions. It is NOT an incrementing number.

If SDL wasn't built from a git repository with the appropriate tools, this will return an empty string.

You shouldn't use this function for anything but logging it for debugging purposes. The string is not intended to be reliable in any way.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetVersion

SDL_GetRGB

Get RGB values from a pixel in the specified format.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
void SDL_GetRGB(Uint32 pixel,
              const SDL_PixelFormat * format,
              Uint8 * r, Uint8 * g, Uint8 * b);
```

Function Parameters

pixel	a pixel value
format	an SDL_PixelFormat structure describing the format of the pixel
r	a pointer filled in with the red component
g	a pointer filled in with the green component
b	a pointer filled in with the blue component

Remarks

This function uses the entire 8-bit [0..255] range when converting color components from pixel formats with less than 8-bits per RGB component (e.g., a completely white pixel in 16-bit RGB565 format would return [0xff, 0xff, 0xff] not [0xf8, 0xfc, 0xf8]).

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRGBA
- SDL_MapRGB
- SDL_MapRGBA

SDL_GetRGBA

Get RGBA values from a pixel in the specified format.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
void SDL_GetRGBA(Uint32 pixel,
                 const SDL_PixelFormat * format,
                 Uint8 * r, Uint8 * g, Uint8 * b,
                 Uint8 * a);
```

Function Parameters

pixel	a pixel value
format	an SDL_PixelFormat structure describing the format of the pixel
r	a pointer filled in with the red component
g	a pointer filled in with the green component
b	a pointer filled in with the blue component
a	a pointer filled in with the alpha component

Remarks

This function uses the entire 8-bit [0..255] range when converting color components from pixel formats with less than 8-bits per RGB component (e.g., a completely white pixel in 16-bit RGB565 format would return [0xff, 0xff, 0xff] not [0xf8, 0xfc, 0xf8]).

If the surface has no alpha component, the alpha will be returned as 0xff (100

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRGB
- SDL_MapRGB
- SDL_MapRGBA

SDL_GetScancodeFromKey

Get the scancode corresponding to the given key code according to the current keyboard layout.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_Scancode SDL_GetScancodeFromKey(SDL_Keycode key);
```

Function Parameters

key	the desired SDL_Keycode to query

Return Value

Returns the SDL_Scancode that corresponds to the given SDL_Keycode.

Remarks

See SDL_Scancode for details.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyFromScancode
- SDL_GetScancodeName

SDL_GetScancodeFromName

Get a scancode from a human-readable name.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_Scancode SDL_GetScancodeFromName(const char *name);
```

Function Parameters

name	the human-readable scancode name

Return Value

Returns the SDL_Scancode, or `SDL_SCANCODE_UNKNOWN` if the name wasn't recognized; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyFromName
- SDL_GetScancodeFromKey
- SDL_GetScancodeName

SDL_GetScancodeName

Get a human-readable name for a scancode.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
const char* SDL_GetScancodeName(SDL_Scancode scancode);
```

Function Parameters

scancode	the desired SDL_Scancode to query

Return Value

Returns a pointer to the name for the scancode. If the scancode doesn't have a name this function returns an empty string ("").

Remarks

See SDL_Scancode for details. **Warning**: The returned name is by design not stable across platforms, e.g. the name for `SDL_SCANCODE_LGUI` is "Left GUI" under Linux but "Left Windows" under Microsoft Windows, and some scancodes like `SDL_SCANCODE_NONUSBACKSLASH` don't have any name at all. There are even scancodes that share names, e.g. `SDL_SCANCODE_RETURN` and `SDL_SCANCODE_RETURN2` (both called "Return"). This function is therefore unsuitable for creating a stable cross-platform two-way mapping between strings and scancodes.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetScancodeFromKey
- SDL_GetScancodeFromName

SDL_GetSemaphoreValue

Get the current value of a semaphore.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
Uint32 SDL_GetSemaphoreValue(SDL_Semaphore *sem);
```

Function Parameters

sem	the semaphore to query

Return Value

Returns the current value of the semaphore.

Version

This function is available since SDL 3.0.0.

SDL_GetSensorData

Get the current state of an opened sensor.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
int SDL_GetSensorData(SDL_Sensor *sensor, float *data, int num_values);
```

Function Parameters

sensor	The SDL_Sensor object to query
data	A pointer filled with the current sensor state
num_values	The number of values to write to data

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The number of values and interpretation of the data is sensor dependent.

Version

This function is available since SDL 3.0.0.

SDL_GetSensorFromInstanceID

Return the SDL_Sensor associated with an instance ID.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
SDL_Sensor* SDL_GetSensorFromInstanceID(SDL_SensorID instance_id);
```

Function Parameters

instance_id	the sensor instance ID

Return Value

Returns an SDL_Sensor object.

Version

This function is available since SDL 3.0.0.

SDL_GetSensorInstanceID

Get the instance ID of a sensor.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
SDL_SensorID SDL_GetSensorInstanceID(SDL_Sensor *sensor);
```

Function Parameters

sensor	The SDL_Sensor object to inspect

Return Value

Returns the sensor instance ID, or 0 if `sensor` is NULL.

Version

This function is available since SDL 3.0.0.

SDL_GetSensorInstanceName

Get the implementation dependent name of a sensor.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
const char* SDL_GetSensorInstanceName(SDL_SensorID instance_id);
```

Function Parameters

instance_id	the sensor instance ID

Return Value

Returns the sensor name, or NULL if `instance_id` is not valid

Version

This function is available since SDL 3.0.0.

SDL_GetSensorInstanceNonPortableType

Get the platform dependent type of a sensor.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
int SDL_GetSensorInstanceNonPortableType(SDL_SensorID instance_id);
```

Function Parameters

instance_id	the sensor instance ID

Return Value

Returns the sensor platform dependent type, or -1 if `instance_id` is not valid

Version

This function is available since SDL 3.0.0.

SDL_GetSensorInstanceType

Get the type of a sensor.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
SDL_SensorType SDL_GetSensorInstanceType(SDL_SensorID instance_id);
```

Function Parameters

instance_id	the sensor instance ID

Return Value

Returns the SDL_SensorType, or `SDL_SENSOR_INVALID` if `instance_id` is not valid

Version

This function is available since SDL 3.0.0.

SDL_GetSensorName

Get the implementation dependent name of a sensor.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
const char* SDL_GetSensorName(SDL_Sensor *sensor);
```

Function Parameters

sensor	The SDL_Sensor object

Return Value

Returns the sensor name, or NULL if `sensor` is NULL.

Version

This function is available since SDL 3.0.0.

SDL_GetSensorNonPortableType

Get the platform dependent type of a sensor.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
int SDL_GetSensorNonPortableType(SDL_Sensor *sensor);
```

Function Parameters

sensor	The SDL_Sensor object to inspect

Return Value

Returns the sensor platform dependent type, or -1 if `sensor` is NULL.

Version

This function is available since SDL 3.0.0.

SDL_GetSensorProperties

Get the properties associated with a sensor.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
SDL_PropertiesID SDL_GetSensorProperties(SDL_Sensor *sensor);
```

Function Parameters

sensor	The SDL_Sensor object

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetSensors

Get a list of currently connected sensors.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
SDL_SensorID* SDL_GetSensors(int *count);
```

Function Parameters

count	a pointer filled in with the number of sensors returned

Return Value

Returns a 0 terminated array of sensor instance IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Version

This function is available since SDL 3.0.0.

SDL_GetSensorType

Get the type of a sensor.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
SDL_SensorType SDL_GetSensorType(SDL_Sensor *sensor);
```

Function Parameters

sensor	The SDL_Sensor object to inspect

Return Value

Returns the SDL_SensorType type, or `SDL_SENSOR_INVALID` if `sensor` is NULL.

Version

This function is available since SDL 3.0.0.

SDL_GetSilenceValueForFormat

Get the appropriate memset value for silencing an audio format.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_GetSilenceValueForFormat(SDL_AudioFormat format);
```

Function Parameters

format	the audio data format to query.

Return Value

Returns A byte value that can be passed to memset.

Remarks

The value returned by this function can be used as the second argument to memset (or SDL_memset) to set an audio buffer in a specific format to silence.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_GetStorageFileSize

Query the size of a file within a storage container.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
int SDL_GetStorageFileSize(SDL_Storage *storage, const char *path,
    Uint64 *length);
```

Function Parameters

storage	a storage container to query
path	the relative path of the file to query
length	a pointer to be filled with the file's length

Return Value

Returns 0 if the file could be queried, a negative value otherwise; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ReadStorageFile
- SDL_StorageReady

SDL_GetStoragePathInfo

Get information about a filesystem path in a storage container.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
int SDL_GetStoragePathInfo(SDL_Storage *storage, const char *path,
    SDL_PathInfo *info);
```

Function Parameters

storage	a storage container
path	the path to query
info	a pointer filled in with information about the path, or NULL to check for the existence of a file

Return Value

Returns 0 on success or a negative error code if the file doesn't exist, or another failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StorageReady

SDL_GetStorageSpaceRemaining

Queries the remaining space in a storage container.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
Uint64 SDL_GetStorageSpaceRemaining(SDL_Storage *storage);
```

Function Parameters

storage	a storage container to query

Return Value

Returns the amount of remaining space, in bytes

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StorageReady
- SDL_WriteStorageFile

SDL_GetStringProperty

Get a string property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
const char* SDL_GetStringProperty(SDL_PropertiesID props, const char
    *name, const char *default_value);
```

Function Parameters

props	the properties to query
name	the name of the property to query
default_value	the default value of the property

Return Value

Returns the value of the property, or `default_value` if it is not set or not a string property.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPropertyType
- SDL_HasProperty
- SDL_SetStringProperty

SDL_GetSurfaceAlphaMod

Get the additional alpha value used in blit operations.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_GetSurfaceAlphaMod(SDL_Surface *surface, Uint8 *alpha);
```

Function Parameters

surface	the SDL_Surface structure to query
alpha	a pointer filled in with the current alpha value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetSurfaceColorMod
- SDL_SetSurfaceAlphaMod

SDL_GetSurfaceBlendMode

Get the blend mode used for blit operations.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_GetSurfaceBlendMode(SDL_Surface *surface, SDL_BlendMode
    *blendMode);
```

Function Parameters

surface	the SDL_Surface structure to query
blendMode	a pointer filled in with the current SDL_BlendMode

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetSurfaceBlendMode

SDL_GetSurfaceClipRect

Get the clipping rectangle for a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_GetSurfaceClipRect(SDL_Surface *surface, SDL_Rect *rect);
```

Function Parameters

surface	the SDL_Surface structure representing the surface to be clipped
rect	an SDL_Rect structure filled in with the clipping rectangle for the surface

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

When `surface` is the destination of a blit, only the area within the clip rectangle is drawn into.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetSurfaceClipRect

SDL_GetSurfaceColorKey

Get the color key (transparent pixel) for a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_GetSurfaceColorKey(SDL_Surface *surface, Uint32 *key);
```

Function Parameters

surface	the SDL_Surface structure to query
key	a pointer filled in with the transparent pixel

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The color key is a pixel of the format used by the surface, as generated by SDL_MapRGB().

If the surface doesn't have color key enabled this function returns -1.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetSurfaceColorKey
- SDL_SurfaceHasColorKey

SDL_GetSurfaceColorMod

Get the additional color value multiplied into blit operations.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_GetSurfaceColorMod(SDL_Surface *surface, Uint8 *r, Uint8 *g,
    Uint8 *b);
```

Function Parameters

surface	the SDL_Surface structure to query
r	a pointer filled in with the current red color value
g	a pointer filled in with the current green color value
b	a pointer filled in with the current blue color value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetSurfaceAlphaMod
- SDL_SetSurfaceColorMod

SDL_GetSurfaceColorspace

Get the colorspace used by a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_GetSurfaceColorspace(SDL_Surface *surface, SDL_Colorspace
    *colorspace);
```

Function Parameters

surface	the SDL_Surface structure to query
colorspace	a pointer filled in with an SDL_ColorSpace value describing the surface colorspace

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The colorspace defaults to SDL_COLORSPACE_SRGB_LINEAR for floating point formats, SDL_COLORSPACE_HDR10 for 10-bit formats, SDL_COLORSPACE_SRGB for other RGB surfaces and SDL_COLORSPACE_BT709_FULL for YUV textures.

Version

This function is available since SDL 3.0.0.

SDL_GetSurfaceProperties

Get the properties associated with a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_PropertiesID SDL_GetSurfaceProperties(SDL_Surface *surface);
```

Function Parameters

surface	the SDL_Surface structure to query

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Remarks

The following properties are understood by SDL:

- `SDL_PROP_SURFACE_COLORSPACE_NUMBER`: an SDL_ColorSpace value describing the surface colorspace, defaults to SDL_COLORSPACE_SRGB_LINEAR for floating point formats, SDL_COLORSPACE_HDR10 for 10-bit formats, SDL_COLORSPACE_SRGB for other RGB surfaces and SDL_COLORSPACE_BT709_FULL for YUV surfaces.
- `SDL_PROP_SURFACE_SDR_WHITE_POINT_FLOAT`: for HDR10 and floating point surfaces, this defines the value of 100diffuse white, with higher values being displayed in the High Dynamic Range headroom. This defaults to 203 for HDR10 surfaces and 1.0 for floating point surfaces.
- `SDL_PROP_SURFACE_HDR_HEADROOM_FLOAT`: for HDR10 and floating point surfaces, this defines the maximum dynamic range used by the content, in terms of the SDR white point. This defaults to 0.0, which disables tone mapping.
- `SDL_PROP_SURFACE_TONEMAP_OPERATOR_STRING`: the tone mapping operator used when compressing from a surface with high dynamic range to another with lower dynamic range. Currently this supports "chrome", which uses the same tone mapping that Chrome uses for HDR content,

the form "*=N", where N is a floating point scale factor applied in linear space, and "none", which disables tone mapping. This defaults to "chrome".

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetSystemRAM

Get the amount of RAM configured in the system.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
int SDL_GetSystemRAM(void);
```

Return Value

Returns the amount of RAM configured in the system in MiB.

Version

This function is available since SDL 3.0.0.

SDL_GetSystemTheme

Get the current system theme.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_SystemTheme SDL_GetSystemTheme(void);
```

Return Value

Returns the current system theme, light, dark, or unknown

Version

This function is available since SDL 3.0.0.

SDL_GetTextureAlphaMod

Get the additional alpha value multiplied into render copy operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetTextureAlphaMod(SDL_Texture *texture, Uint8 *alpha);
```

Function Parameters

texture	the texture to query
alpha	a pointer filled in with the current alpha value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureAlphaModFloat
- SDL_GetTextureColorMod
- SDL_SetTextureAlphaMod

SDL_GetTextureAlphaModFloat

Get the additional alpha value multiplied into render copy operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetTextureAlphaModFloat(SDL_Texture *texture, float *alpha);
```

Function Parameters

texture	the texture to query
alpha	a pointer filled in with the current alpha value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureAlphaMod
- SDL_GetTextureColorModFloat
- SDL_SetTextureAlphaModFloat

SDL_GetTextureBlendMode

Get the blend mode used for texture copy operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetTextureBlendMode(SDL_Texture *texture, SDL_BlendMode
    *blendMode);
```

Function Parameters

texture	the texture to query
blendMode	a pointer filled in with the current SDL_BlendMode

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetTextureBlendMode

SDL_GetTextureColorMod

Get the additional color value multiplied into render copy operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetTextureColorMod(SDL_Texture *texture, Uint8 *r, Uint8 *g,
    Uint8 *b);
```

Function Parameters

texture	the texture to query
r	a pointer filled in with the current red color value
g	a pointer filled in with the current green color value
b	a pointer filled in with the current blue color value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureAlphaMod
- SDL_GetTextureColorModFloat
- SDL_SetTextureColorMod

SDL_GetTextureColorModFloat

Get the additional color value multiplied into render copy operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetTextureColorModFloat(SDL_Texture *texture, float *r, float
    *g, float *b);
```

Function Parameters

texture	the texture to query
r	a pointer filled in with the current red color value
g	a pointer filled in with the current green color value
b	a pointer filled in with the current blue color value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureAlphaModFloat
- SDL_GetTextureColorMod
- SDL_SetTextureColorModFloat

SDL_GetTextureProperties

Get the properties associated with a texture.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_PropertiesID SDL_GetTextureProperties(SDL_Texture *texture);
```

Function Parameters

texture	the texture to query

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Remarks

The following read-only properties are provided by SDL:

- `SDL_PROP_TEXTURE_COLORSPACE_NUMBER`: an SDL_ColorSpace value describing the colorspace used by the texture
- `SDL_PROP_TEXTURE_SDR_WHITE_POINT_FLOAT`: for HDR10 and floating point textures, this defines the value of 100diffuse white, with higher values being displayed in the High Dynamic Range headroom. This defaults to 100 for HDR10 textures and 1.0 for other textures.
- `SDL_PROP_TEXTURE_HDR_HEADROOM_FLOAT`: for HDR10 and floating point textures, this defines the maximum dynamic range used by the content, in terms of the SDR white point. If this is defined, any values outside the range supported by the display will be scaled into the available HDR headroom, otherwise they are clipped. This defaults to 1.0 for SDR textures, 4.0 for HDR10 textures, and no default for floating point textures.

With the direct3d11 renderer:

- `SDL_PROP_TEXTURE_D3D11_TEXTURE_POINTER`: the ID3D11Texture2D associated with the texture
- `SDL_PROP_TEXTURE_D3D11_TEXTURE_U_POINTER`: the ID3D11Texture2D associated with the U plane of a YUV texture

- SDL_PROP_TEXTURE_D3D11_TEXTURE_V_POINTER: the ID3D11Texture2D associated with the V plane of a YUV texture

With the direct3d12 renderer:

- SDL_PROP_TEXTURE_D3D12_TEXTURE_POINTER: the ID3D12Resource associated with the texture
- SDL_PROP_TEXTURE_D3D12_TEXTURE_U_POINTER: the ID3D12Resource associated with the U plane of a YUV texture
- SDL_PROP_TEXTURE_D3D12_TEXTURE_V_POINTER: the ID3D12Resource associated with the V plane of a YUV texture

With the vulkan renderer:

- SDL_PROP_TEXTURE_VULKAN_TEXTURE_POINTER: the VkImage associated with the texture
- SDL_PROP_TEXTURE_VULKAN_TEXTURE_U_POINTER: the VkImage associated with the U plane of a YUV texture
- SDL_PROP_TEXTURE_VULKAN_TEXTURE_V_POINTER: the VkImage associated with the V plane of a YUV texture
- SDL_PROP_TEXTURE_VULKAN_TEXTURE_UV_POINTER: the VkImage associated with the UV plane of a NV12/NV21 texture

With the opengl renderer:

- SDL_PROP_TEXTURE_OPENGL_TEXTURE_NUMBER: the GLuint texture associated with the texture
- SDL_PROP_TEXTURE_OPENGL_TEXTURE_UV_NUMBER: the GLuint texture associated with the UV plane of an NV12 texture
- SDL_PROP_TEXTURE_OPENGL_TEXTURE_U_NUMBER: the GLuint texture associated with the U plane of a YUV texture
- SDL_PROP_TEXTURE_OPENGL_TEXTURE_V_NUMBER: the GLuint texture associated with the V plane of a YUV texture
- SDL_PROP_TEXTURE_OPENGL_TEXTURE_TARGET_NUMBER: the GLenum for the texture target (GL_TEXTURE_2D, GL_TEXTURE_RECTANGLE_ARB, etc)
- SDL_PROP_TEXTURE_OPENGL_TEX_W_FLOAT: the texture coordinate width of the texture (0.0 - 1.0)
- SDL_PROP_TEXTURE_OPENGL_TEX_H_FLOAT: the texture coordinate height of the texture (0.0 - 1.0)

With the opengles2 renderer:

- `SDL_PROP_TEXTURE_OPENGLES2_TEXTURE_NUMBER`: the GLuint texture associated with the texture
- `SDL_PROP_TEXTURE_OPENGLES2_TEXTURE_UV_NUMBER`: the GLuint texture associated with the UV plane of an NV12 texture
- `SDL_PROP_TEXTURE_OPENGLES2_TEXTURE_U_NUMBER`: the GLuint texture associated with the U plane of a YUV texture
- `SDL_PROP_TEXTURE_OPENGLES2_TEXTURE_V_NUMBER`: the GLuint texture associated with the V plane of a YUV texture
- `SDL_PROP_TEXTURE_OPENGLES2_TEXTURE_TARGET_NUMBER`: the GLenum for the texture target (`GL_TEXTURE_2D`, `GL_TEXTURE_EXTERNAL_OES`, etc)

With the vulkan renderer:

- `SDL_PROP_TEXTURE_VULKAN_TEXTURE_NUMBER`: the VkImage associated with the texture

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_GetTextureScaleMode

Get the scale mode used for texture scale operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_GetTextureScaleMode(SDL_Texture *texture, SDL_ScaleMode
    *scaleMode);
```

Function Parameters

texture	the texture to query.
scaleMode	a pointer filled in with the current scale mode.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetTextureScaleMode

SDL_GetThreadID

Get the thread identifier for the specified thread.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
SDL_ThreadID SDL_GetThreadID(SDL_Thread * thread);
```

Function Parameters

thread	the thread to query

Return Value

Returns the ID of the specified thread, or the ID of the current thread if `thread` is NULL.

Remarks

This thread identifier is as reported by the underlying operating system. If SDL is running on a platform that does not support threads the return value will always be zero.

Code Examples

```
#include <SDL3/SDL.h>
#include <stdlib.h>

// Very simple thread - counts 0 to 9 delaying 50ms between increments
int TestThread(void *ptr)
{
    int cnt;

    for (cnt = 0; cnt < 10; ++cnt) {
        SDL_Log("Thread counter: %d", cnt);
        SDL_Delay(50);
    }

    return cnt;
}
```

```
int main(int argc, char *argv[])
{
    SDL_Thread  *thread;
    SDL_ThreadID threadID;
    int         threadReturnValue;

    SDL_Log("Simple SDL_CreateThread test:");

    /* Simply create a thread */
    thread = SDL_CreateThread(TestThread, "TestThread", (void *)NULL);

    if (NULL == thread) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "SDL_CreateThread
            failed: %s\n", SDL_GetError());
        exit(-1);
    }

    /* Retrieve the ID for the newly launched thread */
    threadID = SDL_GetThreadID(thread);

    /* Wait for the thread to complete and get the return code */
    SDL_WaitThread(thread, &threadReturnValue);
    SDL_Log("Thread returned value: %d", threadReturnValue);

    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetCurrentThreadID

SDL_GetThreadName

Get the thread name as it was specified in SDL_CreateThread().

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
const char* SDL_GetThreadName(SDL_Thread *thread);
```

Function Parameters

thread	the thread to query

Return Value

Returns a pointer to a UTF-8 string that names the specified thread, or NULL if it doesn't have a name.

Remarks

This is internal memory, not to be freed by the caller, and remains valid until the specified thread is cleaned up by SDL_WaitThread().

Version

This function is available since SDL 3.0.0.

SDL_GetTicks

Get the number of milliseconds since SDL library initialization.

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
Uint64 SDL_GetTicks(void);
```

Return Value

Returns an unsigned 64-bit value representing the number of milliseconds since the SDL library initialized.

Code Examples

```
int variable;
SDL_bool quit = SDL_FALSE;
unsigned int lastTime = 0, currentTime;
while (!quit) {
  // do stuff
  // ...

  // Print a report once per second
  currentTime = SDL_GetTicks();
  if (currentTime > lastTime + 1000) {
    printf("Report: %d\n", variable);
    lastTime = currentTime;
  }
}
```

Version

This function is available since SDL 3.0.0.

SDL_GetTicksNS

Get the number of nanoseconds since SDL library initialization.

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
Uint64 SDL_GetTicksNS(void);
```

Return Value

Returns an unsigned 64-bit value representing the number of nanoseconds since the SDL library initialized.

Version

This function is available since SDL 3.0.0.

SDL_GetTLS

Get the current thread's value associated with a thread local storage ID.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
void * SDL_GetTLS(SDL_TLSID id);
```

Function Parameters

id	the thread local storage ID

Return Value

Returns the value associated with the ID for the current thread or NULL if no value has been set; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetTLS

SDL_GetTouchDeviceName

Get the touch device name as reported from the driver.

Header File

Defined in SDL3/SDL_touch.h

Syntax

```
const char* SDL_GetTouchDeviceName(SDL_TouchID touchID);
```

Function Parameters

touchID	the touch device instance ID.

Return Value

Returns touch device name, or NULL on error; call SDL_GetError() for more details.

Remarks

You do not own the returned string, do not modify or free it.

Version

This function is available since SDL 3.0.0.

SDL_GetTouchDevices

Get a list of registered touch devices.

Header File

Defined in SDL3/SDL_touch.h

Syntax

```
SDL_TouchID* SDL_GetTouchDevices(int *count);
```

Function Parameters

count	a pointer filled in with the number of devices returned, can be NULL.

Return Value

Returns a 0 terminated array of touch device IDs which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Remarks

On some platforms SDL first sees the touch device if it was actually used. Therefore the returned list might be empty, although devices are available. After using all devices at least once the number will be correct.

Version

This function is available since SDL 3.0.0.

SDL_GetTouchDeviceType

Get the type of the given touch device.

Header File

Defined in SDL3/SDL_touch.h

Syntax

```
SDL_TouchDeviceType SDL_GetTouchDeviceType(SDL_TouchID touchID);
```

Function Parameters

touchID	the ID of a touch device

Return Value

Returns touch device type

Version

This function is available since SDL 3.0.0.

SDL_GetTouchFingers

Get a list of active fingers for a given touch device.

Header File

Defined in SDL3/SDL_touch.h

Syntax

```
extern DECLSPEC SDL_Finger **SDLCALL SDL_GetTouchFingers(SDL_TouchID
    touchID, int *count);
```

Function Parameters

touchID	the ID of a touch device
count	a pointer filled in with the number of fingers returned, can be NULL.

Return Value

Returns a NULL terminated array of SDL_Finger pointers which should be freed with SDL_free(), or NULL on error; call SDL_GetError() for more details.

Version

This function is available since SDL 3.0.0.

SDL_GetUserFolder

Finds the most suitable user folder for the specified purpose, and returns its path in OS-specific notation.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
char* SDL_GetUserFolder(SDL_Folder folder);
```

Function Parameters

folder	The type of folder to find

Return Value

Returns Either a null-terminated C string containing the full path to the folder, or NULL if an error happened.

Remarks

Many OSes provide certain standard folders for certain purposes, such as storing pictures, music or videos for a certain user. This function gives the path for many of those special locations.

This function is specifically for _user_ folders, which are meant for the user to access and manage. For application-specific folders, meant to hold data for the application to manage, see SDL_GetBasePath() and SDL_GetPrefPath().

Note that the function is expensive, and should be called once at the beginning of the execution and kept for as long as needed.

The returned path is guaranteed to end with a path separator ('
' on Windows, '/' on most other platforms).

The returned value is owned by the caller and should be freed with SDL_free().

If NULL is returned, the error may be obtained with SDL_GetError().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Folder

SDL_GetVersion

Get the version of SDL that is linked against your program.

Header File

Defined in SDL3/SDL_version.h

Syntax

```
int SDL_GetVersion(SDL_Version * ver);
```

Function Parameters

ver	the SDL_Version structure that contains the version information

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If you are linking to SDL dynamically, then it is possible that the current version will be different than the version you compiled against. This function returns the current version, while SDL_VERSION() is a macro that tells you what version you compiled with.

This function may be called safely at any time, even before SDL_Init().

Code Examples

```
SDL_Version compiled;
SDL_Version linked;

SDL_VERSION(&compiled);
SDL_GetVersion(&linked);
SDL_Log("We compiled against SDL version %u.%u.%u ...\n",
       compiled.major, compiled.minor, compiled.patch);
SDL_Log("But we are linking against SDL version %u.%u.%u.\n",
       linked.major, linked.minor, linked.patch);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRevision

SDL_GetVideoDriver

Get the name of a built in video driver.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
const char* SDL_GetVideoDriver(int index);
```

Function Parameters

index	the index of a video driver

Return Value

Returns the name of the video driver with the given **index**.

Remarks

The video drivers are presented in the order in which they are normally checked during initialization.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumVideoDrivers

SDL_GetWindowBordersSize

Get the size of a window's borders (decorations) around the client area.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetWindowBordersSize(SDL_Window *window, int *top, int *left,
    int *bottom, int *right);
```

Function Parameters

window	the window to query the size values of the border (decorations) from
top	pointer to variable for storing the size of the top border; NULL is permitted
left	pointer to variable for storing the size of the left border; NULL is permitted
bottom	pointer to variable for storing the size of the bottom border; NULL is permitted
right	pointer to variable for storing the size of the right border; NULL is permitted

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Note: If this function fails (returns -1), the size values will be initialized to 0, 0, 0, 0 (if a non-NULL pointer is provided), as if the window in question was borderless.

Note: This function may fail on systems where the window has not yet been decorated by the display server (for example, immediately after calling SDL_CreateWindow). It is recommended that you wait at least until the window has been presented and composited, so that the window system has a chance to decorate the window and provide the border dimensions to SDL.

This function also returns -1 if getting the information is not supported.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowSize

SDL_GetWindowDisplayScale

Get the content display scale relative to a window's pixel size.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
float SDL_GetWindowDisplayScale(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the display scale, or 0.0f on failure; call SDL_GetError() for more information.

Remarks

This is a combination of the window pixel density and the display content scale, and is the expected scale for displaying content in this window. For example, if a 3840x2160 window had a display scale of 2.0, the user expects the content to take twice as many pixels and be the same physical size as if it were being displayed in a 1920x1080 window with a display scale of 1.0.

Conceptually this value corresponds to the scale display setting, and is updated when that setting is changed, or the window moves to a display with a different scale setting.

Version

This function is available since SDL 3.0.0.

SDL_GetWindowFlags

Get the window flags.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_WindowFlags SDL_GetWindowFlags(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns a mask of the SDL_WindowFlags associated with `window`

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateWindow
- SDL_HideWindow
- SDL_MaximizeWindow
- SDL_MinimizeWindow
- SDL_SetWindowFullscreen
- SDL_SetWindowMouseGrab
- SDL_ShowWindow

SDL_GetWindowFromID

Get a window from a stored ID.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_Window* SDL_GetWindowFromID(SDL_WindowID id);
```

Function Parameters

id	the ID of the window

Return Value

Returns the window associated with `id` or NULL if it doesn't exist; call SDL_GetError() for more information.

Remarks

The numeric ID is what SDL_WindowEvent references, and is necessary to map these events to specific SDL_Window objects.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowID

SDL_GetWindowFullscreenMode

Query the display mode to use when a window is visible at fullscreen.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
const SDL_DisplayMode* SDL_GetWindowFullscreenMode(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns a pointer to the exclusive fullscreen mode to use or NULL for borderless fullscreen desktop mode

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetWindowFullscreenMode
- SDL_SetWindowFullscreen

SDL_GetWindowICCProfile

Get the raw ICC profile data for the screen the window is currently on.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
void* SDL_GetWindowICCProfile(SDL_Window *window, size_t *size);
```

Function Parameters

window	the window to query
size	the size of the ICC profile

Return Value

Returns the raw ICC profile data on success or NULL on failure; call SDL_GetError() for more information.

Remarks

Data returned should be freed with SDL_free.

Version

This function is available since SDL 3.0.0.

SDL_GetWindowID

Get the numeric ID of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_WindowID SDL_GetWindowID(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the ID of the window on success or 0 on failure; call SDL_GetError() for more information.

Remarks

The numeric ID is what SDL_WindowEvent references, and is necessary to map these events to specific SDL_Window objects.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowFromID

SDL_GetWindowKeyboardGrab

Get a window's keyboard grab mode.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_bool SDL_GetWindowKeyboardGrab(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns SDL_TRUE if keyboard is grabbed, and SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetWindowKeyboardGrab

SDL_GetWindowMaximumSize

Get the maximum size of a window's client area.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetWindowMaximumSize(SDL_Window *window, int *w, int *h);
```

Function Parameters

window	the window to query
w	a pointer filled in with the maximum width of the window, may be NULL
h	a pointer filled in with the maximum height of the window, may be NULL

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowMinimumSize
- SDL_SetWindowMaximumSize

SDL_GetWindowMinimumSize

Get the minimum size of a window's client area.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetWindowMinimumSize(SDL_Window *window, int *w, int *h);
```

Function Parameters

window	the window to query
w	a pointer filled in with the minimum width of the window, may be NULL
h	a pointer filled in with the minimum height of the window, may be NULL

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowMaximumSize
- SDL_SetWindowMinimumSize

SDL_GetWindowMouseGrab

Get a window's mouse grab mode.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_bool SDL_GetWindowMouseGrab(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns SDL_TRUE if mouse is grabbed, and SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetWindowKeyboardGrab

SDL_GetWindowMouseRect

Get the mouse confinement rectangle of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
const SDL_Rect* SDL_GetWindowMouseRect(SDL_Window *window);
```

Function Parameters

window	The window to query

Return Value

Returns A pointer to the mouse confinement rectangle of a window, or NULL if there isn't one.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetWindowMouseRect

SDL_GetWindowOpacity

Get the opacity of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetWindowOpacity(SDL_Window *window, float *out_opacity);
```

Function Parameters

window	the window to get the current opacity value from
out_opacity	the float filled in (0.0f - transparent, 1.0f - opaque)

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If transparency isn't supported on this platform, opacity will be reported as 1.0f without error.

The parameter `opacity` is ignored if it is NULL.

This function also returns -1 if an invalid window was provided.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetWindowOpacity

SDL_GetWindowParent

Get parent of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_Window* SDL_GetWindowParent(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the parent of the window on success or NULL if the window has no parent.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreatePopupWindow

SDL_GetWindowPixelDensity

Get the pixel density of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
float SDL_GetWindowPixelDensity(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the pixel density or 0.0f on failure; call SDL_GetError() for more information.

Remarks

This is a ratio of pixel size to window size. For example, if the window is 1920x1080 and it has a high density back buffer of 3840x2160 pixels, it would have a pixel density of 2.0.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowDisplayScale

SDL_GetWindowPixelFormat

Get the pixel format associated with the window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
Uint32 SDL_GetWindowPixelFormat(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the pixel format of the window on success or SDL_PIXELFORMAT_UNKNOWN on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GetWindowPosition

Get the position of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetWindowPosition(SDL_Window *window, int *x, int *y);
```

Function Parameters

window	the window to query
x	a pointer filled in with the x position of the window, may be NULL
y	a pointer filled in with the y position of the window, may be NULL

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This is the current position of the window as last reported by the windowing system.

If you do not need the value for one of the positions a NULL may be passed in the `x` or `y` parameter.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetWindowPosition

SDL_GetWindowProperties

Get the properties associated with a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_PropertiesID SDL_GetWindowProperties(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns a valid property ID on success or 0 on failure; call SDL_GetError() for more information.

Remarks

The following read-only properties are provided by SDL:

- `SDL_PROP_WINDOW_SHAPE_POINTER`: the surface associated with a shaped window

On Android:

- `SDL_PROP_WINDOW_ANDROID_WINDOW_POINTER`: the ANativeWindow associated with the window
- `SDL_PROP_WINDOW_ANDROID_SURFACE_POINTER`: the EGLSurface associated with the window

On iOS:

- `SDL_PROP_WINDOW_UIKIT_WINDOW_POINTER`: the `(__unsafe_unretained)` UIWindow associated with the window
- `SDL_PROP_WINDOW_UIKIT_METAL_VIEW_TAG_NUMBER`: the NSInteger tag assocated with metal views on the window

On KMS/DRM:

- `SDL_PROP_WINDOW_KMSDRM_DEVICE_INDEX_NUMBER`: the device index associated with the window (e.g. the X in /dev/dri/cardX)

- SDL_PROP_WINDOW_KMSDRM_DRM_FD_NUMBER: the DRM FD associated with the window
- SDL_PROP_WINDOW_KMSDRM_GBM_DEVICE_POINTER: the GBM device associated with the window

On macOS:

- SDL_PROP_WINDOW_COCOA_WINDOW_POINTER: the (__unsafe_unretained) NSWindow associated with the window
- SDL_PROP_WINDOW_COCOA_METAL_VIEW_TAG_NUMBER: the NSInteger tag assocated with metal views on the window

On Vivante:

- SDL_PROP_WINDOW_VIVANTE_DISPLAY_POINTER: the EGLNativeDisplayType associated with the window
- SDL_PROP_WINDOW_VIVANTE_WINDOW_POINTER: the EGLNativeWindowType associated with the window
- SDL_PROP_WINDOW_VIVANTE_SURFACE_POINTER: the EGLSurface associated with the window

On UWP:

- SDL_PROP_WINDOW_WINRT_WINDOW_POINTER: the IInspectable CoreWindow associated with the window

On Windows:

- SDL_PROP_WINDOW_WIN32_HWND_POINTER: the HWND associated with the window
- SDL_PROP_WINDOW_WIN32_HDC_POINTER: the HDC associated with the window
- SDL_PROP_WINDOW_WIN32_INSTANCE_POINTER: the HINSTANCE associated with the window

On Wayland:

Note: The xdg_* window objects do not internally persist across window show/hide calls. They will be null if the window is hidden and must be queried each time it is shown.

- SDL_PROP_WINDOW_WAYLAND_DISPLAY_POINTER: the wl_display associated with the window
- SDL_PROP_WINDOW_WAYLAND_SURFACE_POINTER: the wl_surface associated with the window

- `SDL_PROP_WINDOW_WAYLAND_EGL_WINDOW_POINTER`: the wl_egl_window associated with the window
- `SDL_PROP_WINDOW_WAYLAND_XDG_SURFACE_POINTER`: the xdg_surface associated with the window
- `SDL_PROP_WINDOW_WAYLAND_XDG_TOPLEVEL_POINTER`: the xdg_toplevel role associated with the window
- 'SDL_PROP_WINDOW_WAYLAND_XDG_TOPLEVEL_EXPORT_HANDLE_STRING': the export handle associated with the window
- `SDL_PROP_WINDOW_WAYLAND_XDG_POPUP_POINTER`: the xdg_popup role associated with the window
- `SDL_PROP_WINDOW_WAYLAND_XDG_POSITIONER_POINTER`: the xdg_positioner associated with the window, in popup mode

On X11:

- `SDL_PROP_WINDOW_X11_DISPLAY_POINTER`: the X11 Display associated with the window
- `SDL_PROP_WINDOW_X11_SCREEN_NUMBER`: the screen number associated with the window
- `SDL_PROP_WINDOW_X11_WINDOW_NUMBER`: the X11 Window associated with the window

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_GetNumberProperty

SDL_GetWindowSize

Get the size of a window's client area.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetWindowSize(SDL_Window *window, int *w, int *h);
```

Function Parameters

window	the window to query the width and height from
w	a pointer filled in with the width of the window, may be NULL
h	a pointer filled in with the height of the window, may be NULL

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

NULL can safely be passed as the `w` or `h` parameter if the width or height value is not desired.

The window pixel size may differ from its window coordinate size if the window is on a high pixel density display. Use SDL_GetWindowSizeInPixels() or SDL_GetRenderOutputSize() to get the real client area size in pixels.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderOutputSize
- SDL_GetWindowSizeInPixels
- SDL_SetWindowSize

SDL_GetWindowSizeInPixels

Get the size of a window's client area, in pixels.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GetWindowSizeInPixels(SDL_Window *window, int *w, int *h);
```

Function Parameters

window	the window from which the drawable size should be queried
w	a pointer to variable for storing the width in pixels, may be NULL
h	a pointer to variable for storing the height in pixels, may be NULL

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateWindow
- SDL_GetWindowSize

SDL_GetWindowSurface

Get the SDL surface associated with the window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_Surface* SDL_GetWindowSurface(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the surface associated with the window, or NULL on failure; call SDL_GetError() for more information.

Remarks

A new surface will be created with the optimal format for the window, if necessary. This surface will be freed when the window is destroyed. Do not free this surface.

This surface will be invalidated if the window is resized. After resizing a window this function must be called again to return a valid surface.

You may not combine this with 3D or the rendering API on this window.

This function is affected by `SDL_HINT_FRAMEBUFFER_ACCELERATION`.

Code Examples

```
#include <SDL3/SDL.h> // include SDL header

int main(int argc, char* argv[])
{
    SDL_Surface *screen; // even with SDL2, we can still bring ancient
        code back
    SDL_Window *window;
    SDL_Surface *image;

    SDL_Init(SDL_INIT_VIDEO); // init video

    // create the window like normal
```

```
    window = SDL_CreateWindow("SDL2 Example", 640, 480, 0);

    // but instead of creating a renderer, we can draw directly to the
        screen
    screen = SDL_GetWindowSurface(window);

    // let's just show some classic code for reference
    image = SDL_LoadBMP("box.bmp"); // loads image
    SDL_BlitSurface(image, NULL, screen, NULL); // blit it to the screen
    SDL_DestroySurface(image);

    // this works just like SDL_Flip() in SDL 1.2
    SDL_UpdateWindowSurface(window);

    // show image for 2 seconds
    SDL_Delay(2000);
    SDL_DestroyWindow(window);
    SDL_Quit();
    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroyWindowSurface
- SDL_WindowHasSurface
- SDL_UpdateWindowSurface
- SDL_UpdateWindowSurfaceRects

SDL_GetWindowTitle

Get the title of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
const char* SDL_GetWindowTitle(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns the title of the window in UTF-8 format or "" if there is no title.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetWindowTitle

SDL_GL_CreateContext

Create an OpenGL context for an OpenGL window, and make it current.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_GLContext SDL_GL_CreateContext(SDL_Window *window);
```

Function Parameters

window	the window to associate with the context

Return Value

Returns the OpenGL context associated with `window` or NULL on error; call SDL_GetError() for more details.

Remarks

Windows users new to OpenGL should note that, for historical reasons, GL functions added after OpenGL version 1.1 are not available by default. Those functions must be loaded at run-time, either with an OpenGL extension-handling library or with SDL_GL_GetProcAddress() and its related functions.

SDL_GLContext is an alias for `void *`. It's opaque to the application.

Code Examples

```
// Window mode MUST include SDL_WINDOW_OPENGL for use with OpenGL.
SDL_Window *window = SDL_CreateWindow(
    "SDL2/OpenGL Demo", 640, 480,
    SDL_WINDOW_OPENGL|SDL_WINDOW_RESIZABLE);

// Create an OpenGL context associated with the window.
SDL_GLContext glcontext = SDL_GL_CreateContext(window);

// now you can make GL calls.
glClearColor(0,0,0,1);
glClear(GL_COLOR_BUFFER_BIT);
SDL_GL_SwapWindow(window);

// Once finished with OpenGL functions, the SDL_GLContext can be deleted.
```

```
SDL_GL_DeleteContext(glcontext);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_DeleteContext
- SDL_GL_MakeCurrent

SDL_GL_DeleteContext

Delete an OpenGL context.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GL_DeleteContext(SDL_GLContext context);
```

Function Parameters

context	the OpenGL context to be deleted

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_CreateContext

SDL_GL_ExtensionSupported

Check if an OpenGL extension is supported for the current context.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_bool SDL_GL_ExtensionSupported(const char *extension);
```

Function Parameters

extension	the name of the extension to check

Return Value

Returns SDL_TRUE if the extension is supported, SDL_FALSE otherwise.

Remarks

This function operates on the current GL context; you must have created a context and it must be current before calling this function. Do not assume that all contexts you create will have the same set of extensions available, or that recreating an existing context will offer the same extensions again.

While it's probably not a massive overhead, this function is not an O(1) operation. Check the extensions you care about after creating the GL context and save that information somewhere instead of calling the function every time you need to know.

Code Examples

```
extern void draw_to_the_screen_with_framebuffer_blit(void);
extern void draw_to_the_screen_with_a_textured_quad(void);

if (SDL_GL_ExtensionSupported("GL_EXT_framebuffer_blit")) {
    draw_to_the_screen_with_framebuffer_blit(); // faster!
} else {
    draw_to_the_screen_with_a_textured_quad(); // slower!
}
```

Version

This function is available since SDL 3.0.0.

SDL_GL_GetAttribute

Get the actual value for an attribute from the current context.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GL_GetAttribute(SDL_GLattr attr, int *value);
```

Function Parameters

attr	an SDL_GLattr enum value specifying the OpenGL attribute to get
value	a pointer filled in with the current value of `attr`

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_ResetAttributes
- SDL_GL_SetAttribute

SDL_GL_GetCurrentContext

Get the currently active OpenGL context.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_GLContext SDL_GL_GetCurrentContext(void);
```

Return Value

Returns the currently active OpenGL context or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_MakeCurrent

SDL_GL_GetCurrentWindow

Get the currently active OpenGL window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_Window* SDL_GL_GetCurrentWindow(void);
```

Return Value

Returns the currently active OpenGL window on success or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_GL_GetProcAddress

Get an OpenGL function by name.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_FunctionPointer SDL_GL_GetProcAddress(const char *proc);
```

Function Parameters

proc	the name of an OpenGL function

Return Value

Returns a pointer to the named OpenGL function. The returned pointer should be cast to the appropriate function signature.

Remarks

If the GL library is loaded at runtime with SDL_GL_LoadLibrary(), then all GL functions must be retrieved this way. Usually this is used to retrieve function pointers to OpenGL extensions.

There are some quirks to looking up OpenGL functions that require some extra care from the application. If you code carefully, you can handle these quirks without any platform-specific code, though:

- On Windows, function pointers are specific to the current GL context; this means you need to have created a GL context and made it current before calling SDL_GL_GetProcAddress(). If you recreate your context or create a second context, you should assume that any existing function pointers aren't valid to use with it. This is (currently) a Windows-specific limitation, and in practice lots of drivers don't suffer this limitation, but it is still the way the wgl API is documented to work and you should expect crashes if you don't respect it. Store a copy of the function pointers that comes and goes with context lifespan.

- On X11, function pointers returned by this function are valid for any context, and can even be looked up before a context is created at all. This means that, for at least some common OpenGL implementations, if you look up a function that doesn't exist, you'll get a non-NULL result that is _NOT_ safe to call. You must always make sure the function is actually

available for a given GL context before calling it, by checking for the existence of the appropriate extension with SDL_GL_ExtensionSupported(), or verifying that the version of OpenGL you're using offers the function as core functionality.

- Some OpenGL drivers, on all platforms, *will* return NULL if a function isn't supported, but you can't count on this behavior. Check for extensions you use, and if you get a NULL anyway, act as if that extension wasn't available. This is probably a bug in the driver, but you can code defensively for this scenario anyhow.
- Just because you're on Linux/Unix, don't assume you'll be using X11. Next-gen display servers are waiting to replace it, and may or may not make the same promises about function pointers.
- OpenGL function pointers must be declared `APIENTRY` as in the example code. This will ensure the proper calling convention is followed on platforms where this matters (Win32) thereby avoiding stack corruption.

Code Examples

```
typedef void (APIENTRY * GL_ActiveTextureARB_Func)(unsigned int);
GL_ActiveTextureARB_Func glActiveTextureARB_ptr = 0;

/* Get function pointer */
glActiveTextureARB_ptr=(GL_ActiveTextureARB_Func)
    SDL_GL_GetProcAddress("glActiveTextureARB");

/* It was your responsibility to make sure this was a valid function to
    call! */
glActiveTextureARB_ptr(GL_TEXTURE0_ARB);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_ExtensionSupported
- SDL_GL_LoadLibrary
- SDL_GL_UnloadLibrary

SDL_GL_GetSwapInterval

Get the swap interval for the current OpenGL context.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GL_GetSwapInterval(int *interval);
```

Function Parameters

interval	Output interval value. 0 if there is no vertical retrace synchronization, 1 if the buffer swap is synchronized with the vertical retrace, and -1 if late swaps happen immediately instead of waiting for the next retrace

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If the system can't determine the swap interval, or there isn't a valid current context, this function will set *interval to 0 as a safe default.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_SetSwapInterval

SDL_GL_LoadLibrary

Dynamically load an OpenGL library.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GL_LoadLibrary(const char *path);
```

Function Parameters

path	the platform dependent OpenGL library name, or NULL to open the default OpenGL library

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This should be done after initializing the video driver, but before creating any OpenGL windows. If no OpenGL library is loaded, the default library will be loaded upon creation of the first OpenGL window.

If you do this, you need to retrieve all of the GL functions used in your program from the dynamic library using SDL_GL_GetProcAddress().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_GetProcAddress
- SDL_GL_UnloadLibrary

SDL_GL_MakeCurrent

Set up an OpenGL context for rendering into an OpenGL window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GL_MakeCurrent(SDL_Window *window, SDL_GLContext context);
```

Function Parameters

window	the window to associate with the context
context	the OpenGL context to associate with the window

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The context must have been created with a compatible window.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_CreateContext

SDL_GL_ResetAttributes

Reset all previously set OpenGL context attributes to their default values.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
void SDL_GL_ResetAttributes(void);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_GetAttribute
- SDL_GL_SetAttribute

SDL_GL_SetAttribute

Set an OpenGL window attribute before window creation.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GL_SetAttribute(SDL_GLattr attr, int value);
```

Function Parameters

attr	an SDL_GLattr enum value specifying the OpenGL attribute to set
value	the desired value for the attribute

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function sets the OpenGL attribute `attr` to `value`. The requested attributes should be set before creating an OpenGL window. You should use SDL_GL_GetAttribute() to check the values after creating the OpenGL context, since the values obtained can differ from the requested ones.

Code Examples

```
SDL_Window *window;
SDL_GLContext context;

SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 5);
SDL_GL_SetAttribute(SDL_GL_GREEN_SIZE, 5);
SDL_GL_SetAttribute(SDL_GL_BLUE_SIZE, 5);
SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 16);
SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);

window = SDL_CreateWindow("OpenGL Window", 640, 480, SDL_WINDOW_OPENGL);
if (!window) {
    fprintf(stderr, "Couldn't create window: %s\n", SDL_GetError());
    return 1;
```

```
}

context = SDL_GL_CreateContext(window);
if (!context) {
    fprintf(stderr, "Couldn't create context: %s\n", SDL_GetError());
    return 1;
}

int r, g, b;
SDL_GL_GetAttribute(SDL_GL_RED_SIZE, &r);
SDL_GL_GetAttribute(SDL_GL_GREEN_SIZE, &g);
SDL_GL_GetAttribute(SDL_GL_BLUE_SIZE, &b);

printf("Red size: %d, Green size: %d, Blue size: %d\n", r, g, b);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_GetAttribute
- SDL_GL_ResetAttributes

SDL_GL_SetSwapInterval

Set the swap interval for the current OpenGL context.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GL_SetSwapInterval(int interval);
```

Function Parameters

interval	0 for immediate updates, 1 for updates synchronized with the vertical retrace, -1 for adaptive vsync

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Some systems allow specifying -1 for the interval, to enable adaptive vsync. Adaptive vsync works the same as vsync, but if you've already missed the vertical retrace for a given frame, it swaps buffers immediately, which might be less jarring for the user during occasional framerate drops. If an application requests adaptive vsync and the system does not support it, this function will fail and return -1. In such a case, you should probably retry the call with 1 for the interval.

Adaptive vsync is implemented for some glX drivers with GLX_EXT_swap_control_tear, and for some Windows drivers with WGL_EXT_swap_control_tear.

Read more on the Khronos wiki: https://www.khronos.org/opengl/wiki/Swap_Interval#Adaptive_Vsync

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_GetSwapInterval

SDL_GL_SwapWindow

Update a window with OpenGL rendering.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_GL_SwapWindow(SDL_Window *window);
```

Function Parameters

window	the window to change

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This is used with double-buffered OpenGL contexts, which are the default.

On macOS, make sure you bind 0 to the draw framebuffer before swapping the window, otherwise nothing will happen. If you aren't using glBindFramebuffer(), this is the default and you won't have to do anything extra.

Code Examples

```
SDL_Window* window = SDL_CreateWindow("SDL3/OpenGL Demo", 640, 480,
    SDL_WINDOW_OPENGL|SDL_WINDOW_RESIZABLE);

/* Create an OpenGL context associated with the window. */
SDL_GLContext glcontext = SDL_GL_CreateContext(window);

/* This makes our buffer swap syncronized with the monitor's vertical
    refresh */
SDL_GL_SetSwapInterval(1);

/* Clear context */
glClearColor(0,0,0,1);
glClear(GL_COLOR_BUFFER_BIT);

/* <Extra drawing functions here> */
```

```
/* Swap our buffer to display the current contents of buffer on screen */
SDL_GL_SwapWindow(window);
```

Version

This function is available since SDL 3.0.0.

SDL_GL_UnloadLibrary

Unload the OpenGL library previously loaded by SDL_GL_LoadLibrary().

Header File

Defined in SDL3/SDL_video.h

Syntax

```
void SDL_GL_UnloadLibrary(void);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GL_LoadLibrary

SDL_GlobDirectory

Enumerate a directory tree, filtered by pattern, and return a list.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
extern DECLSPEC char **SDLCALL SDL_GlobDirectory(const char *path, const
    char *pattern, Uint32 flags, int *count);
```

Function Parameters

path	the path of the directory to enumerate
pattern	the pattern that files in the directory must match. Can be NULL.
flags	`SDL_GLOB_*` bitflags that affect this search.
count	on return, will be set to the number of items in the returned array. Can be NULL.

Return Value

Returns an array of strings on success or NULL on failure; call SDL_GetError() for more information. The caller should pass the returned pointer to SDL_free when done with it.

Remarks

Files are filtered out if they don't match the string in `pattern`, which may contain wildcard characters *' ' (match everything) and '?' (match one character). If pattern is NULL, no filtering is done and all results are returned. Subdirectories are permitted, and are specified with a path separator of '/'. Wildcard characters* '' and '?' never match a path separator. `flags` may be set to SDL_GLOB_CASEINSENSITIVE to make the pattern matching case-insensitive.

The returned array is always NULL-terminated, for your iterating convenience, but if `count` is non-NULL, on return it will contain the number of items in the array, not counting the NULL terminator.

You must free the returned pointer with SDL_free() when done with it.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_GlobStorageDirectory

Enumerate a directory tree, filtered by pattern, and return a list.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
extern DECLSPEC char **SDLCALL SDL_GlobStorageDirectory(SDL_Storage
    *storage, const char *path, const char *pattern, Uint32 flags, int
    *count);
```

Function Parameters

storage	a storage container
path	the path of the directory to enumerate
pattern	the pattern that files in the directory must match. Can be NULL.
flags	`SDL_GLOB_*` bitflags that affect this search.
count	on return, will be set to the number of items in the returned array. Can be NULL.

Return Value

Returns an array of strings on success or NULL on failure; call SDL_GetError() for more information. The caller should pass the returned pointer to SDL_free when done with it.

Remarks

Files are filtered out if they don't match the string in `pattern`, which may contain wildcard characters *' ' (match everything) and '?' (match one character). If pattern is NULL, no filtering is done and all results are returned. Subdirectories are permitted, and are specified with a path separator of '/'. Wildcard characters* '' and '?' never match a path separator. `flags` may be set to SDL_GLOB_CASEINSENSITIVE to make the pattern matching case-insensitive.

The returned array is always NULL-terminated, for your iterating convenience, but if `count` is non-NULL, on return it will contain the number of items in the array, not counting the NULL terminator.

You must free the returned pointer with SDL_free() when done with it.

Thread Safety

It is safe to call this function from any thread, assuming the `storage` object is thread-safe.

Version

This function is available since SDL 3.0.0.

SDL_GUIDFromString

Convert a GUID string into a ::SDL_GUID structure.

Header File

Defined in SDL3/SDL_guid.h

Syntax

```
SDL_GUID SDL_GUIDFromString(const char *pchGUID);
```

Function Parameters

pchGUID	string containing an ASCII representation of a GUID

Return Value

Returns a ::SDL_GUID structure.

Remarks

Performs no error checking. If this function is given a string containing an invalid GUID, the function will silently succeed, but the GUID generated will not be useful.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GUIDToString

SDL_GUIDToString

Get an ASCII string representation for a given ::SDL_GUID.

Header File

Defined in SDL3/SDL_guid.h

Syntax

```
int SDL_GUIDToString(SDL_GUID guid, char *pszGUID, int cbGUID);
```

Function Parameters

guid	the ::SDL_GUID you wish to convert to string
pszGUID	buffer in which to write the ASCII string
cbGUID	the size of pszGUID

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

You should supply at least 33 bytes for pszGUID.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GUIDFromString

SDL_HapticEffectSupported

Check to see if an effect is supported by a haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_bool SDL_HapticEffectSupported(SDL_Haptic *haptic, const
    SDL_HapticEffect *effect);
```

Function Parameters

haptic	the SDL_Haptic device to query
effect	the desired effect to query

Return Value

Returns SDL_TRUE if the effect is supported or SDL_FALSE if it isn't.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateHapticEffect
- SDL_GetHapticFeatures

SDL_HapticRumbleSupported

Check whether rumble is supported on a haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_bool SDL_HapticRumbleSupported(SDL_Haptic *haptic);
```

Function Parameters

haptic	haptic device to check for rumble support

Return Value

Returns SDL_TRUE if the effect is supported or SDL_FALSE if it isn't.

Code Examples

```
extern SDL_Haptic* dev;
if (SDL_HapticRumbleSupported(dev)) {
    SDL_InitHapticRumble(dev);
    SDL_PlayHapticRumble(dev, 1.0f, 3000);
    SDL_Delay(3000);
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_InitHapticRumble

SDL_HasAltiVec

Determine whether the CPU has AltiVec features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasAltiVec(void);
```

Return Value

Returns SDL_TRUE if the CPU has AltiVec features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using PowerPC instruction sets.

Version

This function is available since SDL 3.0.0.

SDL_HasARMSIMD

Determine whether the CPU has ARM SIMD (ARMv6) features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasARMSIMD(void);
```

Return Value

Returns SDL_TRUE if the CPU has ARM SIMD features or SDL_FALSE if not.

Remarks

This is different from ARM NEON, which is a different instruction set.

This always returns false on CPUs that aren't using ARM instruction sets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasNEON

SDL_HasAVX

Determine whether the CPU has AVX features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasAVX(void);
```

Return Value

Returns SDL_TRUE if the CPU has AVX features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using Intel instruction sets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasAVX2
- SDL_HasAVX512F

SDL_HasAVX2

Determine whether the CPU has AVX2 features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasAVX2(void);
```

Return Value

Returns SDL_TRUE if the CPU has AVX2 features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using Intel instruction sets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasAVX
- SDL_HasAVX512F

SDL_HasAVX512F

Determine whether the CPU has AVX-512F (foundation) features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasAVX512F(void);
```

Return Value

Returns SDL_TRUE if the CPU has AVX-512F features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using Intel instruction sets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasAVX
- SDL_HasAVX2

SDL_HasClipboardData

Query whether there is data in the clipboard for the provided mime type.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
SDL_bool SDL_HasClipboardData(const char *mime_type);
```

Function Parameters

mime_type	The mime type to check for data for

Return Value

Returns SDL_TRUE if there exists data in clipboard for the provided mime type, SDL_FALSE if it does not.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetClipboardData
- SDL_GetClipboardData

SDL_HasClipboardText

Query whether the clipboard exists and contains a non-empty text string.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
SDL_bool SDL_HasClipboardText(void);
```

Return Value

Returns SDL_TRUE if the clipboard has text, or SDL_FALSE if it does not.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetClipboardText
- SDL_SetClipboardText

SDL_HasEvent

Check for the existence of a certain event type in the event queue.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
SDL_bool SDL_HasEvent(Uint32 type);
```

Function Parameters

type	the type of event to be queried; see SDL_EventType for details

Return Value

Returns SDL_TRUE if events matching `type` are present, or SDL_FALSE if events matching `type` are not present.

Remarks

If you need to check for a range of event types, use SDL_HasEvents() instead.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasEvents

SDL_HasEvents

Check for the existence of certain event types in the event queue.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
SDL_bool SDL_HasEvents(Uint32 minType, Uint32 maxType);
```

Function Parameters

minType	the low end of event type to be queried, inclusive; see SDL_EventType for details
maxType	the high end of event type to be queried, inclusive; see SDL_EventType for details

Return Value

Returns SDL_TRUE if events with type ¿= `minType` and ¡= `maxType` are present, or SDL_FALSE if not.

Remarks

If you need to check for a single event type, use SDL_HasEvent() instead.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasEvents

SDL_HasExactlyOneBitSet32

Determine if a unsigned 32-bit value has exactly one bit set.

Header File

Defined in SDL3/SDL_bits.h

Syntax

```
SDL_FORCE_INLINE SDL_bool SDL_HasExactlyOneBitSet32(Uint32 x);
```

Function Parameters

x	the 32-bit value to examine

Return Value

Returns SDL_TRUE if exactly one bit is set in `x`, SDL_FALSE otherwise.

Remarks

If there are no bits set (`x` is zero), or more than one bit set, this returns SDL_FALSE. If any one bit is exclusively set, this returns SDL_TRUE.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_HasGamepad

Return whether a gamepad is currently connected.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_bool SDL_HasGamepad(void);
```

Return Value

Returns SDL_TRUE if a gamepad is connected, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepads

SDL_HasJoystick

Return whether a joystick is currently connected.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_bool SDL_HasJoystick(void);
```

Return Value

Returns SDL_TRUE if a joystick is connected, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoysticks

SDL_HasKeyboard

Return whether a keyboard is currently connected.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_bool SDL_HasKeyboard(void);
```

Return Value

Returns SDL_TRUE if a keyboard is connected, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyboards

SDL_HasLASX

Determine whether the CPU has LASX (LOONGARCH SIMD) features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasLASX(void);
```

Return Value

Returns SDL_TRUE if the CPU has LOONGARCH LASX features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using LOONGARCH instruction sets.

Version

This function is available since SDL 3.0.0.

SDL_HasLSX

Determine whether the CPU has LSX (LOONGARCH SIMD) features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasLSX(void);
```

Return Value

Returns SDL_TRUE if the CPU has LOONGARCH LSX features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using LOONGARCH instruction sets.

Version

This function is available since SDL 3.0.0.

SDL_HasMMX

Determine whether the CPU has MMX features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasMMX(void);
```

Return Value

Returns SDL_TRUE if the CPU has MMX features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using Intel instruction sets.

Version

This function is available since SDL 3.0.0.

SDL_HasMouse

Return whether a mouse is currently connected.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
SDL_bool SDL_HasMouse(void);
```

Return Value

Returns SDL_TRUE if a mouse is connected, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetMice

SDL_HasNEON

Determine whether the CPU has NEON (ARM SIMD) features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasNEON(void);
```

Return Value

Returns SDL_TRUE if the CPU has ARM NEON features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using ARM instruction sets.

Version

This function is available since SDL 3.0.0.

SDL_HasPrimarySelectionText

Query whether the primary selection exists and contains a non-empty text string.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
SDL_bool SDL_HasPrimarySelectionText(void);
```

Return Value

Returns SDL_TRUE if the primary selection has text, or SDL_FALSE if it does not.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPrimarySelectionText
- SDL_SetPrimarySelectionText

SDL_HasProperty

Return whether a property exists in a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
SDL_bool SDL_HasProperty(SDL_PropertiesID props, const char *name);
```

Function Parameters

props	the properties to query
name	the name of the property to query

Return Value

Returns SDL_TRUE if the property exists, or SDL_FALSE if it doesn't.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPropertyType

SDL_HasRectIntersection

Determine whether two rectangles intersect.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_bool SDL_HasRectIntersection(const SDL_Rect * A,
                                 const SDL_Rect * B);
```

Function Parameters

A	an SDL_Rect structure representing the first rectangle
B	an SDL_Rect structure representing the second rectangle

Return Value

Returns SDL_TRUE if there is an intersection, SDL_FALSE otherwise.

Remarks

If either pointer is NULL the function will return SDL_FALSE.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRectIntersection

SDL_HasRectIntersectionFloat

Determine whether two rectangles intersect with float precision.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_bool SDL_HasRectIntersectionFloat(const SDL_FRect * A,
                                      const SDL_FRect * B);
```

Function Parameters

A	an SDL_FRect structure representing the first rectangle
B	an SDL_FRect structure representing the second rectangle

Return Value

Returns SDL_TRUE if there is an intersection, SDL_FALSE otherwise.

Remarks

If either pointer is NULL the function will return SDL_FALSE.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRectIntersection

SDL_HasScreenKeyboardSupport

Check whether the platform has screen keyboard support.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_bool SDL_HasScreenKeyboardSupport(void);
```

Return Value

Returns SDL_TRUE if the platform has some screen keyboard support or SDL_FALSE if not.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StartTextInput
- SDL_ScreenKeyboardShown

SDL_HasSSE

Determine whether the CPU has SSE features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasSSE(void);
```

Return Value

Returns SDL_TRUE if the CPU has SSE features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using Intel instruction sets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasSSE2
- SDL_HasSSE3
- SDL_HasSSE41
- SDL_HasSSE42

SDL_HasSSE2

Determine whether the CPU has SSE2 features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasSSE2(void);
```

Return Value

Returns SDL_TRUE if the CPU has SSE2 features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using Intel instruction sets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasSSE
- SDL_HasSSE3
- SDL_HasSSE41
- SDL_HasSSE42

SDL_HasSSE3

Determine whether the CPU has SSE3 features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasSSE3(void);
```

Return Value

Returns SDL_TRUE if the CPU has SSE3 features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using Intel instruction sets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasSSE
- SDL_HasSSE2
- SDL_HasSSE41
- SDL_HasSSE42

SDL_HasSSE41

Determine whether the CPU has SSE4.1 features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasSSE41(void);
```

Return Value

Returns SDL_TRUE if the CPU has SSE4.1 features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using Intel instruction sets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasSSE
- SDL_HasSSE2
- SDL_HasSSE3
- SDL_HasSSE42

SDL_HasSSE42

Determine whether the CPU has SSE4.2 features.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
SDL_bool SDL_HasSSE42(void);
```

Return Value

Returns SDL_TRUE if the CPU has SSE4.2 features or SDL_FALSE if not.

Remarks

This always returns false on CPUs that aren't using Intel instruction sets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasSSE
- SDL_HasSSE2
- SDL_HasSSE3
- SDL_HasSSE41

SDL_hid_ble_scan

Start or stop a BLE scan on iOS and tvOS to pair Steam Controllers.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
void SDL_hid_ble_scan(SDL_bool active);
```

Function Parameters

active	SDL_TRUE to start the scan, SDL_FALSE to stop the scan

Version

This function is available since SDL 3.0.0.

SDL_hid_close

Close a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_close(SDL_hid_device *dev);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_hid_device_change_count

Check to see if devices may have been added or removed.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
Uint32 SDL_hid_device_change_count(void);
```

Return Value

Returns a change counter that is incremented with each potential device change, or 0 if device change detection isn't available.

Remarks

Enumerating the HID devices is an expensive operation, so you can call this to see if there have been any system device changes since the last call to this function. A change in the counter returned doesn't necessarily mean that anything has changed, but you can call SDL_hid_enumerate() to get an updated device list.

Calling this function for the first time may cause a thread or other system resource to be allocated to track device change notifications.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_hid_enumerate

SDL_hid_enumerate

Enumerate the HID Devices.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
SDL_hid_device_info * SDL_hid_enumerate(unsigned short vendor_id,
    unsigned short product_id);
```

Function Parameters

vendor_id	The Vendor ID (VID) of the types of device to open, or 0 to match any vendor.
product_id	The Product ID (PID) of the types of device to open, or 0 to match any product.

Return Value

Returns a pointer to a linked list of type SDL_hid_device_info, containing information about the HID devices attached to the system, or NULL in the case of failure. Free this linked list by calling SDL_hid_free_enumeration().

Remarks

This function returns a linked list of all the HID devices attached to the system which match vendor_id and product_id. If `vendor_id` is set to 0 then any vendor matches. If `product_id` is set to 0 then any product matches. If `vendor_id` and `product_id` are both set to 0, then all HID devices will be returned.

By default SDL will only enumerate controllers, to reduce risk of hanging or crashing on bad drivers, but SDL_HINT_HIDAPI_ENUMERATE_ONLY_CONTROLLERS can be set to "0" to enumerate all HID devices.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_hid_device_change_count

SDL_hid_exit

Finalize the HIDAPI library.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_exit(void);
```

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function frees all of the static data associated with HIDAPI. It should be called at the end of execution to avoid memory leaks.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_hid_init

SDL_hid_free_enumeration

Free an enumeration linked list.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
void SDL_hid_free_enumeration(SDL_hid_device_info *devs);
```

Function Parameters

devs	Pointer to a list of struct_device returned from SDL_hid_enumerate().

Remarks

This function frees a linked list created by SDL_hid_enumerate().

Version

This function is available since SDL 3.0.0.

SDL_hid_get_device_info

Get the device info from a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
SDL_hid_device_info * SDL_hid_get_device_info(SDL_hid_device *dev);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().

Return Value

Returns a pointer to the SDL_hid_device_info for this hid_device, or NULL in the case of failure; call SDL_GetError() for more information. This struct is valid until the device is closed with SDL_hid_close().

Version

This function is available since SDL 3.0.0.

SDL_hid_get_feature_report

Get a feature report from a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_get_feature_report(SDL_hid_device *dev, unsigned char *data,
    size_t length);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
data	A buffer to put the read data into, including the Report ID. Set the first byte of `data` to the Report ID of the report to be read, or set it to zero if your device does not use numbered reports.
length	The number of bytes to read, including an extra byte for the report ID. The buffer can be longer than the actual report.

Return Value

Returns the number of bytes read plus one for the report ID (which is still in the first byte), or -1 on error.

Remarks

Set the first byte of `data` to the Report ID of the report to be read. Make sure to allow space for this extra byte in `data`. Upon return, the first byte will still contain the Report ID, and the report data will start in data[1].

Version

This function is available since SDL 3.0.0.

SDL_hid_get_indexed_string

Get a string from a HID device, based on its string index.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_get_indexed_string(SDL_hid_device *dev, int string_index,
    wchar_t *string, size_t maxlen);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
string_index	The index of the string to get.
string	A wide string buffer to put the data into.
maxlen	The length of the buffer in multiples of wchar_t.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_hid_get_input_report

Get an input report from a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_get_input_report(SDL_hid_device *dev, unsigned char *data,
    size_t length);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
data	A buffer to put the read data into, including the Report ID. Set the first byte of `data` to the Report ID of the report to be read, or set it to zero if your device does not use numbered reports.
length	The number of bytes to read, including an extra byte for the report ID. The buffer can be longer than the actual report.

Return Value

Returns the number of bytes read plus one for the report ID (which is still in the first byte), or -1 on error.

Remarks

Set the first byte of `data` to the Report ID of the report to be read. Make sure to allow space for this extra byte in `data`. Upon return, the first byte will still contain the Report ID, and the report data will start in data[1].

Version

This function is available since SDL 3.0.0.

SDL_hid_get_manufacturer_string

Get The Manufacturer String from a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_get_manufacturer_string(SDL_hid_device *dev, wchar_t
    *string, size_t maxlen);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
string	A wide string buffer to put the data into.
maxlen	The length of the buffer in multiples of wchar_t.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_hid_get_product_string

Get The Product String from a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_get_product_string(SDL_hid_device *dev, wchar_t *string,
    size_t maxlen);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
string	A wide string buffer to put the data into.
maxlen	The length of the buffer in multiples of wchar_t.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_hid_get_report_descriptor

Get a report descriptor from a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_get_report_descriptor(SDL_hid_device *dev, unsigned char
    *buf, size_t buf_size);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
buf	The buffer to copy descriptor into.
buf_size	The size of the buffer in bytes.

Return Value

Returns the number of bytes actually copied, or -1 on error; call SDL_GetError() for more information.

Remarks

User has to provide a preallocated buffer where descriptor will be copied to. The recommended size for a preallocated buffer is 4096 bytes.

Version

This function is available since SDL 3.0.0.

SDL_hid_get_serial_number_string

Get The Serial Number String from a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_get_serial_number_string(SDL_hid_device *dev, wchar_t
    *string, size_t maxlen);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
string	A wide string buffer to put the data into.
maxlen	The length of the buffer in multiples of wchar_t.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_hid_init

Initialize the HIDAPI library.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_init(void);
```

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function initializes the HIDAPI library. Calling it is not strictly necessary, as it will be called automatically by SDL_hid_enumerate() and any of the SDL_hid_open_*() functions if it is needed. This function should be called at the beginning of execution however, if there is a chance of HIDAPI handles being opened by different threads simultaneously.

Each call to this function should have a matching call to SDL_hid_exit()

Version

This function is available since SDL 3.0.0.

See Also

- SDL_hid_exit

SDL_hid_open

Open a HID device using a Vendor ID (VID), Product ID (PID) and optionally a serial number.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
SDL_hid_device * SDL_hid_open(unsigned short vendor_id, unsigned short
    product_id, const wchar_t *serial_number);
```

Function Parameters

vendor_id	The Vendor ID (VID) of the device to open.
product_id	The Product ID (PID) of the device to open.
serial_number	The Serial Number of the device to open (Optionally NULL).

Return Value

Returns a pointer to a SDL_hid_device object on success or NULL on failure.

Remarks

If `serial_number` is NULL, the first device with the specified VID and PID is opened.

Version

This function is available since SDL 3.0.0.

SDL_hid_open_path

Open a HID device by its path name.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
SDL_hid_device * SDL_hid_open_path(const char *path);
```

Function Parameters

path	The path name of the device to open

Return Value

Returns a pointer to a SDL_hid_device object on success or NULL on failure.

Remarks

The path name be determined by calling SDL_hid_enumerate(), or a platform-specific path name can be used (eg: /dev/hidraw0 on Linux).

Version

This function is available since SDL 3.0.0.

SDL_hid_read

Read an Input report from a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_read(SDL_hid_device *dev, unsigned char *data, size_t
    length);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
data	A buffer to put the read data into.
length	The number of bytes to read. For devices with multiple reports, make sure to read an extra byte for the report number.

Return Value

Returns the actual number of bytes read and -1 on error. If no packet was available to be read and the handle is in non-blocking mode, this function returns 0.

Remarks

Input reports are returned to the host through the INTERRUPT IN endpoint. The first byte will contain the Report number if the device uses numbered reports.

Version

This function is available since SDL 3.0.0.

SDL_hid_read_timeout

Read an Input report from a HID device with timeout.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_read_timeout(SDL_hid_device *dev, unsigned char *data,
    size_t length, int milliseconds);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
data	A buffer to put the read data into.
length	The number of bytes to read. For devices with multiple reports, make sure to read an extra byte for the report number.
milliseconds	timeout in milliseconds or -1 for blocking wait.

Return Value

Returns the actual number of bytes read and -1 on error. If no packet was available to be read within the timeout period, this function returns 0.

Remarks

Input reports are returned to the host through the INTERRUPT IN endpoint. The first byte will contain the Report number if the device uses numbered reports.

Version

This function is available since SDL 3.0.0.

SDL_hid_send_feature_report

Send a Feature report to the device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_send_feature_report(SDL_hid_device *dev, const unsigned char
    *data, size_t length);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
data	The data to send, including the report number as the first byte.
length	The length in bytes of the data to send, including the report number.

Return Value

Returns the actual number of bytes written and -1 on error.

Remarks

Feature reports are sent over the Control endpoint as a Set_Report transfer. The first byte of `data` must contain the Report ID. For devices which only support a single report, this must be set to 0x0. The remaining bytes contain the report data. Since the Report ID is mandatory, calls to SDL_hid_send_feature_report() will always contain one more byte than the report contains. For example, if a hid report is 16 bytes long, 17 bytes must be passed to SDL_hid_send_feature_report(): the Report ID (or 0x0, for devices which do not use numbered reports), followed by the report data (16 bytes). In this example, the length passed in would be 17.

Version

This function is available since SDL 3.0.0.

SDL_hid_set_nonblocking

Set the device handle to be non-blocking.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_set_nonblocking(SDL_hid_device *dev, int nonblock);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
nonblock	enable or not the nonblocking reads - 1 to enable nonblocking - 0 to disable nonblocking.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

In non-blocking mode calls to SDL_hid_read() will return immediately with a value of 0 if there is no data to be read. In blocking mode, SDL_hid_read() will wait (block) until there is data to read before returning.

Nonblocking can be turned on and off at any time.

Version

This function is available since SDL 3.0.0.

SDL_hid_write

Write an Output report to a HID device.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
int SDL_hid_write(SDL_hid_device *dev, const unsigned char *data, size_t
    length);
```

Function Parameters

dev	A device handle returned from SDL_hid_open().
data	The data to send, including the report number as the first byte.
length	The length in bytes of the data to send.

Return Value

Returns the actual number of bytes written and -1 on error.

Remarks

The first byte of `data` must contain the Report ID. For devices which only support a single report, this must be set to 0x0. The remaining bytes contain the report data. Since the Report ID is mandatory, calls to SDL_hid_write() will always contain one more byte than the report contains. For example, if a hid report is 16 bytes long, 17 bytes must be passed to SDL_hid_write(), the Report ID (or 0x0, for devices with a single report), followed by the report data (16 bytes). In this example, the length passed in would be 17.

SDL_hid_write() will send the data on the first OUT endpoint, if one exists. If it does not, it will send the data through the Control Endpoint (Endpoint 0).

Version

This function is available since SDL 3.0.0.

SDL_HideCursor

Hide the cursor.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
int SDL_HideCursor(void);
```

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CursorVisible
- SDL_ShowCursor

SDL_HideWindow

Hide a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_HideWindow(SDL_Window *window);
```

Function Parameters

window	the window to hide

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ShowWindow

SDL_iconv_string

This function converts a buffer or string between encodings in one pass, returning a string that must be freed with SDL_free() or NULL on error.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
char* SDL_iconv_string(const char *tocode,
                       const char *fromcode,
                       const char *inbuf,
                       size_t inbytesleft);
```

Version

This function is available since SDL 3.0.0.

SDL_Init

Initialize the SDL library.

Header File

Defined in SDL3/SDL_init.h

Syntax

```
int SDL_Init(Uint32 flags);
```

Function Parameters

flags	subsystem initialization flags

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

SDL_Init() simply forwards to calling SDL_InitSubSystem(). Therefore, the two may be used interchangeably. Though for readability of your code SDL_InitSubSystem() might be preferred.

The file I/O (for example: SDL_IOFromFile) and threading (SDL_CreateThread) subsystems are initialized by default. Message boxes (SDL_ShowSimpleMessageBox) also attempt to work without initializing the video subsystem, in hopes of being useful in showing an error dialog when SDL_Init fails. You must specifically initialize other subsystems if you use them in your application.

Logging (such as SDL_Log) works without initialization, too. `flags` may be any of the following OR'd together:

- `SDL_INIT_TIMER`: timer subsystem
- `SDL_INIT_AUDIO`: audio subsystem
- `SDL_INIT_VIDEO`: video subsystem; automatically initializes the events subsystem
- `SDL_INIT_JOYSTICK`: joystick subsystem; automatically initializes the events subsystem
- `SDL_INIT_HAPTIC`: haptic (force feedback) subsystem

- SDL_INIT_GAMEPAD: gamepad subsystem; automatically initializes the joystick subsystem
- SDL_INIT_EVENTS: events subsystem
- SDL_INIT_SENSOR: sensor subsystem

Subsystem initialization is ref-counted, you must call SDL_QuitSubSystem() for each SDL_InitSubSystem() to correctly shutdown a subsystem manually (or call SDL_Quit() to force shutdown). If a subsystem is already loaded then this call will increase the ref-count and return.

Code Examples

```
#include <SDL3/SDL.h>

int main(int argc, char* argv[])
{
    if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) != 0) {
        SDL_Log("Unable to initialize SDL: %s", SDL_GetError());
        return 1;
    }

    /* ... */

    SDL_Quit();

    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_InitSubSystem
- SDL_Quit
- SDL_SetMainReady
- SDL_WasInit

SDL_InitHapticRumble

Initialize a haptic device for simple rumble playback.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_InitHapticRumble(SDL_Haptic *haptic);
```

Function Parameters

haptic	the haptic device to initialize for simple rumble playback

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PlayHapticRumble
- SDL_StopHapticRumble
- SDL_HapticRumbleSupported

SDL_InitSubSystem

Compatibility function to initialize the SDL library.

Header File

Defined in SDL3/SDL_init.h

Syntax

```
int SDL_InitSubSystem(Uint32 flags);
```

Function Parameters

flags	any of the flags used by SDL_Init(); see SDL_Init for details.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function and SDL_Init() are interchangeable.

Code Examples

```
/* Separating Joystick and Video initialization. */
SDL_Init(SDL_INIT_VIDEO);

SDL_Window* window = SDL_CreateWindow("A Window",
    640, 480,
    SDL_WINDOW_FULLSCREEN);
SDL_Renderer* renderer = SDL_CreateRenderer(window, NULL, 0);

/* Do Some Video stuff */

/* Initialize the joystick subsystem */
SDL_InitSubSystem(SDL_INIT_JOYSTICK);

/* Do some stuff with video and joystick */

/* Shut them both down */
SDL_Quit();
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Init
- SDL_Quit
- SDL_QuitSubSystem

SDL_IOFromConstMem

Use this function to prepare a read-only memory buffer for use with SDL_IOStream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_IOStream* SDL_IOFromConstMem(const void *mem, size_t size);
```

Function Parameters

mem	a pointer to a read-only buffer to feed an SDL_IOStream stream
size	the buffer size, in bytes

Return Value

Returns a pointer to a new SDL_IOStream structure, or NULL if it fails; call SDL_GetError() for more information.

Remarks

This function sets up an SDL_IOStream struct based on a memory area of a certain size. It assumes the memory area is not writable.

Attempting to write to this SDL_IOStream stream will report an error without writing to the memory buffer.

This memory buffer is not copied by the SDL_IOStream; the pointer you provide must remain valid until you close the stream. Closing the stream will not free the original buffer.

If you need to write to a memory buffer, you should use SDL_IOFromMem() with a writable buffer of memory instead.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_IOFromMem
- SDL_CloseIO
- SDL_ReadIO

- SDL_SeekIO
- SDL_TellIO

SDL_IOFromDynamicMem

Use this function to create an SDL_IOStream that is backed by dynamically allocated memory.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_IOStream* SDL_IOFromDynamicMem(void);
```

Return Value

Returns a pointer to a new SDL_IOStream structure, or NULL if it fails; call SDL_GetError() for more information.

Remarks

This supports the following properties to provide access to the memory and control over allocations: - `SDL_PROP_IOSTREAM_DYNAMIC_MEMORY_POINTER`: a pointer to the internal memory of the stream. This can be set to NULL to transfer ownership of the memory to the application, which should free the memory with SDL_free(). If this is done, the next operation on the stream must be SDL_CloseIO(). - `SDL_PROP_IOSTREAM_DYNAMIC_CHUNKSIZE_NUMBER`: memory will be allocated in multiples of this size, defaulting to 1024.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseIO
- SDL_ReadIO
- SDL_SeekIO
- SDL_TellIO
- SDL_WriteIO

SDL_IOFromFile

Use this function to create a new SDL_IOStream structure for reading from and/or writing to a named file.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_IOStream* SDL_IOFromFile(const char *file, const char *mode);
```

Function Parameters

file	a UTF-8 string representing the filename to open
mode	an ASCII string representing the mode to be used for opening the file.

Return Value

Returns a pointer to the SDL_IOStream structure that is created, or NULL on failure; call SDL_GetError() for more information.

Remarks

The `mode` string is treated roughly the same as in a call to the C library's fopen(), even if SDL doesn't happen to use fopen() behind the scenes.

Available `mode` strings:

- "r": Open a file for reading. The file must exist.
- "w": Create an empty file for writing. If a file with the same name already exists its content is erased and the file is treated as a new empty file.
- "a": Append to a file. Writing operations append data at the end of the file. The file is created if it does not exist.
- "r+": Open a file for update both reading and writing. The file must exist.
- "w+": Create an empty file for both reading and writing. If a file with the same name already exists its content is erased and the file is treated as a new empty file.

- "a+": Open a file for reading and appending. All writing operations are performed at the end of the file, protecting the previous content to be overwritten. You can reposition (fseek, rewind) the internal pointer to anywhere in the file for reading, but writing operations will move it back to the end of file. The file is created if it does not exist. **NOTE**: In order to open a file as a binary file, a "b" character has to be included in the `mode` string. This additional "b" character can either be appended at the end of the string (thus making the following compound modes: "rb", "wb", "ab", "r+b", "w+b", "a+b") or be inserted between the letter and the "+" sign for the mixed modes ("rb+", "wb+", "ab+"). Additional characters may follow the sequence, although they should have no effect. For example, "t" is sometimes appended to make explicit the file is a text file.

This function supports Unicode filenames, but they must be encoded in UTF-8 format, regardless of the underlying operating system.

As a fallback, SDL_IOFromFile() will transparently open a matching filename in an Android app's `assets`.

Closing the SDL_IOStream will close SDL's internal file handle.

The following properties may be set at creation time by SDL:

- `SDL_PROP_IOSTREAM_WINDOWS_HANDLE_POINTER`: a pointer, that can be cast to a win32 `HANDLE`, that this SDL_IOStream is using to access the filesystem. If the program isn't running on Windows, or SDL used some other method to access the filesystem, this property will not be set.

- `SDL_PROP_IOSTREAM_STDIO_FILE_POINTER`: a pointer, that can be cast to a stdio `FILE` , that this SDL_IOStream is using to access the filesystem. If SDL used some other method to access the filesystem, this property will not be set. PLEASE NOTE that if SDL is using a different C runtime than your app, trying to use this pointer will almost certainly result in a crash! This is mostly a problem on Windows; make sure you build SDL and your app with the same compiler and settings to avoid it.

- `SDL_PROP_IOSTREAM_ANDROID_AASSET_POINTER`: a pointer, that can be cast to an Android NDK `AAsset` , that this SDL_IOStream is using to access the filesystem. If SDL used some other method to access the filesystem, this property will not be set.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseIO

- SDL_ReadIO
- SDL_SeekIO
- SDL_TellIO
- SDL_WriteIO

SDL_IOFromMem

Use this function to prepare a read-write memory buffer for use with SDL_IOStream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_IOStream* SDL_IOFromMem(void *mem, size_t size);
```

Function Parameters

mem	a pointer to a buffer to feed an SDL_IOStream stream
size	the buffer size, in bytes

Return Value

Returns a pointer to a new SDL_IOStream structure, or NULL if it fails; call SDL_GetError() for more information.

Remarks

This function sets up an SDL_IOStream struct based on a memory area of a certain size, for both read and write access.

This memory buffer is not copied by the SDL_IOStream; the pointer you provide must remain valid until you close the stream. Closing the stream will not free the original buffer.

If you need to make sure the SDL_IOStream never writes to the memory buffer, you should use SDL_IOFromConstMem() with a read-only buffer of memory instead.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_IOFromConstMem
- SDL_CloseIO
- SDL_ReadIO
- SDL_SeekIO

- SDL_TellIO
- SDL_WriteIO

SDL_IOprintf

Print to an SDL_IOStream data stream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
size_t SDL_IOprintf(SDL_IOStream *context, SDL_PRINTF_FORMAT_STRING
    const char *fmt, ...) SDL_PRINTF_VARARG_FUNC(2);
```

Function Parameters

context	a pointer to an SDL_IOStream structure
fmt	a printf() style format string
...	additional parameters matching

Return Value

Returns the number of bytes written, or 0 on error; call SDL_GetError() for more information.

Remarks

This function does formatted printing to the stream.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_IOvprintf
- SDL_WriteIO

SDL_iOSSetAnimationCallback

Use this function to set the animation callback on Apple iOS.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_iOSSetAnimationCallback(SDL_Window * window, int interval, void
    (SDLCALL *callback)(void*), void *callbackParam);
```

Function Parameters

window	the window for which the animation callback should be set
interval	the number of frames after which **callback** will be called
callback	the function to call for every frame.
callbackParam	a pointer that is passed to `callback`.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The function prototype for `callback` is:

```
void callback(void* callbackParam);
```

Where its parameter, `callbackParam`, is what was passed as `callbackParam` to SDL_iOSSetAnimationCallback().

This function is only available on Apple iOS.

For more information see:

https://wiki.libsdl.org/SDL3/README/ios

Note that if you use the "main callbacks" instead of a standard C `main` function, you don't have to use this API, as SDL will manage this for you.

Details on main callbacks are here:

https://wiki.libsdl.org/SDL3/README/main-functions

Version

This function is available since SDL 3.0.0.

See Also

- SDL_iOSSetEventPump

SDL_iOSSetEventPump

Use this function to enable or disable the SDL event pump on Apple iOS.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_iOSSetEventPump(SDL_bool enabled);
```

Function Parameters

enabled	SDL_TRUE to enable the event pump, SDL_FALSE to disable it

Remarks

This function is only available on Apple iOS.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_iOSSetAnimationCallback

SDL_IOvprintf

Print to an SDL_IOStream data stream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
size_t SDL_IOvprintf(SDL_IOStream *context, SDL_PRINTF_FORMAT_STRING
    const char *fmt, va_list ap) SDL_PRINTF_VARARG_FUNCV(2);
```

Function Parameters

context	a pointer to an SDL_IOStream structure
fmt	a printf() style format string
ap	a variable argument list

Return Value

Returns the number of bytes written, or 0 on error; call SDL_GetError() for more information.

Remarks

This function does formatted printing to the stream.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_IOprintf
- SDL_WriteIO

SDL_isalnum

Query if a character is alphabetic (a letter) or a number.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_isalnum(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this will only treat ASCII values for English 'a-z', 'A-Z', and '0-9' as true.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_isalpha

Query if a character is alphabetic (a letter).

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_isalpha(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this will only treat ASCII values for English 'a-z' and 'A-Z' as true.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_IsAndroidTV

Query if the application is running on Android TV.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
SDL_bool SDL_IsAndroidTV(void);
```

Return Value

Returns SDL_TRUE if this is Android TV, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

SDL_isblank

Report if a character is blank (a space or tab).

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_isblank(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this will only treat ASCII values 0x20 (space) or 0x9 (tab) as true.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_IsChromebook

Query if the application is running on a Chromebook.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
SDL_bool SDL_IsChromebook(void);
```

Return Value

Returns SDL_TRUE if this is a Chromebook, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

SDL_iscntrl

Report if a character is a control character.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_iscntrl(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this will only treat ASCII values 0 through 0x1F, and 0x7F, as true.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_IsDeXMode

Query if the application is running on a Samsung DeX docking station.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
SDL_bool SDL_IsDeXMode(void);
```

Return Value

Returns SDL_TRUE if this is a DeX docking station, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

SDL_isdigit

Report if a character is a numeric digit.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_isdigit(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this will only treat ASCII values '0' (0x30) through '9' (0x39), as true.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_IsGamepad

Check if the given joystick is supported by the gamepad interface.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_bool SDL_IsGamepad(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns SDL_TRUE if the given joystick is supported by the gamepad interface, SDL_FALSE if it isn't or it's an invalid index.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoysticks
- SDL_OpenGamepad

SDL_isgraph

Report if a character is any "printable" except space.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_isgraph(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

Be advised that "printable" has a definition that goes back to text terminals from the dawn of computing, making this a sort of special case function that is not suitable for Unicode (or most any) text management. **WARNING**: Regardless of system locale, this is equivalent to `(SDL_isprint(x)) && ((x) != ' ')`.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_isprint

SDL_IsJoystickHaptic

Query if a joystick has haptic features.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_bool SDL_IsJoystickHaptic(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick to test for haptic capabilities

Return Value

Returns SDL_TRUE if the joystick is haptic or SDL_FALSE if it isn't.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenHapticFromJoystick

SDL_IsJoystickVirtual

Query whether or not a joystick is virtual.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_bool SDL_IsJoystickVirtual(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns SDL_TRUE if the joystick is virtual, SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

SDL_islower

Report if a character is lower case.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_islower(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this will only treat ASCII values 'a' through 'z' as true.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_IsMouseHaptic

Query whether or not the current mouse has haptic capabilities.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_bool SDL_IsMouseHaptic(void);
```

Return Value

Returns SDL_TRUE if the mouse is haptic or SDL_FALSE if it isn't.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_OpenHapticFromMouse

SDL_isprint

Report if a character is "printable".

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_isprint(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

Be advised that "printable" has a definition that goes back to text terminals from the dawn of computing, making this a sort of special case function that is not suitable for Unicode (or most any) text management. **WARNING**: Regardless of system locale, this will only treat ASCII values ' ' (0x20) through ' ' (0x7E) as true.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_ispunct

Report if a character is a punctuation mark.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_ispunct(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this is equivalent to `((SDL_isgraph(x)) && (!SDL_isalnum(x)))`.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_isgraph
- SDL_isalnum

SDL_isspace

Report if a character is whitespace.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_isspace(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this will only treat the following ASCII values as true:

- space (0x20)
- tab (0x09)
- newline (0x0A)
- vertical tab (0x0B)
- form feed (0x0C)
- return (0x0D)

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_IsTablet

Query if the current device is a tablet.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
SDL_bool SDL_IsTablet(void);
```

Return Value

Returns SDL_TRUE if the device is a tablet, SDL_FALSE otherwise.

Remarks

If SDL can't determine this, it will return SDL_FALSE.

Version

This function is available since SDL 3.0.0.

SDL_isupper

Report if a character is upper case.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_isupper(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this will only treat ASCII values 'A' through 'Z' as true.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_isxdigit

Report if a character is a hexadecimal digit.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_isxdigit(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns non-zero if x falls within the character class, zero otherwise.

Remarks

WARNING: Regardless of system locale, this will only treat ASCII values 'A' through 'F', 'a' through 'f', and '0' through '9', as true.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_JoystickConnected

Get the status of a specified joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_bool SDL_JoystickConnected(SDL_Joystick *joystick);
```

Function Parameters

joystick	the joystick to query

Return Value

Returns SDL_TRUE if the joystick has been opened, SDL_FALSE if it has not; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_JoystickEventsEnabled

Query the state of joystick event processing.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_bool SDL_JoystickEventsEnabled(void);
```

Return Value

Returns SDL_TRUE if joystick events are being processed, SDL_FALSE otherwise.

Remarks

If joystick events are disabled, you must call SDL_UpdateJoysticks() yourself and check the state of the joystick when you want joystick information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetJoystickEventsEnabled

SDL_LinuxSetThreadPriority

Sets the UNIX nice value for a thread.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_LinuxSetThreadPriority(Sint64 threadID, int priority);
```

Function Parameters

threadID	the Unix thread ID to change priority of.
priority	The new, Unix-specific, priority value.

Return Value

Returns 0 on success, or -1 on error.

Remarks

This uses setpriority() if possible, and RealtimeKit if available.

Version

This function is available since SDL 3.0.0.

SDL_LinuxSetThreadPriorityAndPolicy

Sets the priority (not nice level) and scheduling policy for a thread.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
int SDL_LinuxSetThreadPriorityAndPolicy(Sint64 threadID, int
    sdlPriority, int schedPolicy);
```

Function Parameters

threadID	The Unix thread ID to change priority of.
sdlPriority	The new SDL_ThreadPriority value.
schedPolicy	The new scheduling policy (SCHED_FIFO, SCHED_RR, SCHED_OTHER, etc...)

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This uses setpriority() if possible, and RealtimeKit if available.

Version

This function is available since SDL 3.0.0.

SDL_LoadBMP

Load a BMP image from a file.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_Surface* SDL_LoadBMP(const char *file);
```

Function Parameters

file	the BMP file to load

Return Value

Returns a pointer to a new SDL_Surface structure or NULL if there was an error; call SDL_GetError() for more information.

Remarks

The new surface should be freed with SDL_DestroySurface(). Not doing so will result in a memory leak.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroySurface
- SDL_LoadBMP_IO
- SDL_SaveBMP

SDL_LoadBMP_IO

Load a BMP image from a seekable SDL data stream.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_Surface* SDL_LoadBMP_IO(SDL_IOStream *src, SDL_bool closeio);
```

Function Parameters

src	the data stream for the surface
closeio	if SDL_TRUE, calls SDL_CloseIO() on `src` before returning, even in the case of an error

Return Value

Returns a pointer to a new SDL_Surface structure or NULL if there was an error; call SDL_GetError() for more information.

Remarks

The new surface should be freed with SDL_DestroySurface(). Not doing so will result in a memory leak.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DestroySurface
- SDL_LoadBMP
- SDL_SaveBMP_IO

SDL_LoadFile

Load all the data from a file path.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
void* SDL_LoadFile(const char *file, size_t *datasize);
```

Function Parameters

file	the path to read all available data from
datasize	if not NULL, will store the number of bytes read

Return Value

Returns the data, or NULL if there was an error.

Remarks

The data is allocated with a zero byte at the end (null terminated) for convenience. This extra byte is not included in the value reported via `datasize`.

The data should be freed with SDL_free().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LoadFile_IO

SDL_LoadFile_IO

Load all the data from an SDL data stream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
void* SDL_LoadFile_IO(SDL_IOStream *src, size_t *datasize, SDL_bool
    closeio);
```

Function Parameters

src	the SDL_IOStream to read all available data from
datasize	if not NULL, will store the number of bytes read
closeio	if SDL_TRUE, calls SDL_CloseIO() on `src` before returning, even in the case of an error

Return Value

Returns the data, or NULL if there was an error.

Remarks

The data is allocated with a zero byte at the end (null terminated) for convenience. This extra byte is not included in the value reported via `datasize`.

The data should be freed with SDL_free().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LoadFile

SDL_LoadFunction

Look up the address of the named function in a shared object.

Header File

Defined in SDL3/SDL_loadso.h

Syntax

```
SDL_FunctionPointer SDL_LoadFunction(void *handle, const char *name);
```

Function Parameters

handle	a valid shared object handle returned by SDL_LoadObject()
name	the name of the function to look up

Return Value

Returns a pointer to the function or NULL if there was an error; call SDL_GetError() for more information.

Remarks

This function pointer is no longer valid after calling SDL_UnloadObject().

This function can only look up C function names. Other languages may have name mangling and intrinsic language support that varies from compiler to compiler.

Make sure you declare your function pointers with the same calling convention as the actual library function. Your code will crash mysteriously if you do not do this.

If the requested function doesn't exist, NULL is returned.

Code Examples

```
/* Variable declaration */
void* myHandle = NULL;
const char* myFunctionName = "myFancyFunction";
void (*myFancyFunction)(int anInt);

/* Dynamically load mylib.so */
myHandle = SDL_LoadObject("mylib.so");

/* Load the exported function from mylib.so
 * The exported function has the following prototype
```

```
 * void myFancyFunction(int anInt);
 */
myFancyFunction = (void (*)(int))SDL_LoadFunction(myHandle,
    myFunctionName);

/* Call myFancyFunction with a random integer */
if (myFancyFunction != NULL) {
    myFancyFunction(15);
} else {
    /* Error handling here */
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LoadObject

SDL_LoadObject

Dynamically load a shared object.

Header File

Defined in SDL3/SDL_loadso.h

Syntax

```
void* SDL_LoadObject(const char *sofile);
```

Function Parameters

sofile	a system-dependent name of the object file

Return Value

Returns an opaque pointer to the object handle or NULL if there was an error; call SDL_GetError() for more information.

Code Examples

```
/* Dynamically load mylib.so */
SDL_LoadObject("mylib.so");
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LoadFunction
- SDL_UnloadObject

SDL_LoadWAV

Loads a WAV from a file path.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_LoadWAV(const char *path, SDL_AudioSpec * spec,
              Uint8 ** audio_buf, Uint32 * audio_len);
```

Function Parameters

path	The file path of the WAV file to open.
spec	A pointer to an SDL_AudioSpec that will be set to the WAVE data's format details on successful return.
audio_buf	A pointer filled with the audio data, allocated by the function.
audio_len	A pointer filled with the length of the audio data buffer in bytes

Return Value

Returns 0 on success. `audio_buf` will be filled with a pointer to an allocated buffer containing the audio data, and `audio_len` is filled with the length of that audio buffer in bytes.

This function returns -1 if the .WAV file cannot be opened, uses an unknown data format, or is corrupt; call SDL_GetError() for more information.

When the application is done with the data returned in `audio_buf`, it should call SDL_free() to dispose of it.

Remarks

This is a convenience function that is effectively the same as:

```
SDL_LoadWAV_IO(SDL_IOFromFile(path, "rb"), 1, spec, audio_buf,
    audio_len);
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_free
- SDL_LoadWAV_IO

SDL_LoadWAV_IO

Load the audio data of a WAVE file into memory.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_LoadWAV_IO(SDL_IOStream * src, SDL_bool closeio,
                   SDL_AudioSpec * spec, Uint8 ** audio_buf,
                   Uint32 * audio_len);
```

Function Parameters

src	The data source for the WAVE data
closeio	If SDL_TRUE, calls SDL_CloseIO() on `src` before returning, even in the case of an error
spec	A pointer to an SDL_AudioSpec that will be set to the WAVE data's format details on successful return
audio_buf	A pointer filled with the audio data, allocated by the function
audio_len	A pointer filled with the length of the audio data buffer in bytes

Return Value

Returns 0 on success. `audio_buf` will be filled with a pointer to an allocated buffer containing the audio data, and `audio_len` is filled with the length of that audio buffer in bytes.

This function returns -1 if the .WAV file cannot be opened, uses an unknown data format, or is corrupt; call SDL_GetError() for more information.

When the application is done with the data returned in `audio_buf`, it should call SDL_free() to dispose of it.

Remarks

Loading a WAVE file requires `src`, `spec`, `audio_buf` and `audio_len` to be valid pointers. The entire data portion of the file is then loaded into memory and decoded if necessary.

Supported formats are RIFF WAVE files with the formats PCM (8, 16, 24, and 32 bits), IEEE Float (32 bits), Microsoft ADPCM and IMA ADPCM (4 bits), and A-law and mu-law (8 bits). Other formats are currently unsupported and cause an error.

If this function succeeds, the return value is zero and the pointer to the audio data allocated by the function is written to `audio_buf` and its length in bytes to `audio_len`. The SDL_AudioSpec members `freq`, `channels`, and `format` are set to the values of the audio data in the buffer.

It's necessary to use SDL_free() to free the audio data returned in `audio_buf` when it is no longer used.

Because of the underspecification of the .WAV format, there are many problematic files in the wild that cause issues with strict decoders. To provide compatibility with these files, this decoder is lenient in regards to the truncation of the file, the fact chunk, and the size of the RIFF chunk. The hints `SDL_HINT_WAVE_RIFF_CHUNK_SIZE`, `SDL_HINT_WAVE_TRUNCATION`, and `SDL_HINT_WAVE_FACT_CHUNK` can be used to tune the behavior of the loading process.

Any file that is invalid (due to truncation, corruption, or wrong values in the headers), too big, or unsupported causes an error. Additionally, any critical I/O error from the data source will terminate the loading process with an error. The function returns NULL on error and in all cases (with the exception of `src` being NULL), an appropriate error message will be set.

It is required that the data source supports seeking.

Example:

```
SDL_LoadWAV_IO(SDL_IOFromFile("sample.wav", "rb"), 1, &spec, &buf, &len);
```

Note that the SDL_LoadWAV function does this same thing for you, but in a less messy way:

```
SDL_LoadWAV("sample.wav", &spec, &buf, &len);
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_free
- SDL_LoadWAV

SDL_LockAudioStream

Lock an audio stream for serialized access.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_LockAudioStream(SDL_AudioStream *stream);
```

Function Parameters

stream	The audio stream to lock.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Each SDL_AudioStream has an internal mutex it uses to protect its data structures from threading conflicts. This function allows an app to lock that mutex, which could be useful if registering callbacks on this stream.

One does not need to lock a stream to use in it most cases, as the stream manages this lock internally. However, this lock is held during callbacks, which may run from arbitrary threads at any time, so if an app needs to protect shared data during those callbacks, locking the stream guarantees that the callback is not running while the lock is held.

As this is just a wrapper over SDL_LockMutex for an internal lock; it has all the same attributes (recursive locks are allowed, etc).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_UnlockAudioStream

SDL_LockJoysticks

Locking for atomic access to the joystick API.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
void SDL_LockJoysticks(void) SDL_ACQUIRE(SDL_joystick_lock);
```

Remarks

The SDL joystick functions are thread-safe, however you can lock the joysticks while processing to guarantee that the joystick list won't change and joystick and gamepad events will not be delivered.

Version

This function is available since SDL 3.0.0.

SDL_LockMutex

Lock the mutex.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
void SDL_LockMutex(SDL_Mutex *mutex) SDL_ACQUIRE(mutex);
```

Function Parameters

mutex	the mutex to lock

Remarks

This will block until the mutex is available, which is to say it is in the unlocked state and the OS has chosen the caller as the next thread to lock it. Of all threads waiting to lock the mutex, only one may do so at a time.

It is legal for the owning thread to lock an already-locked mutex. It must unlock it the same number of times before it is actually made available for other threads in the system (this is known as a "recursive mutex").

This function does not fail; if mutex is NULL, it will return immediately having locked nothing. If the mutex is valid, this function will always block until it can lock the mutex, and return with it locked.

Code Examples

¡!– # Begin Mutex Example –¿

```
int status;
SDL_Mutex *mutex;

mutex = SDL_CreateMutex();
if (!mutex) {
  SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create mutex\n");
  return 1;
}

SDL_LockMutex(mutex);

SDL_Log("Locked mutex");

SDL_UnlockMutex(mutex);
```

```
SDL_DestroyMutex(mutex);
```

¡!– # End Mutex Example –¿

Version

This function is available since SDL 3.0.0.

See Also

- SDL_TryLockMutex
- SDL_UnlockMutex

SDL_LockProperties

Lock a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_LockProperties(SDL_PropertiesID props);
```

Function Parameters

props	the properties to lock

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Obtain a multi-threaded lock for these properties. Other threads will wait while trying to lock these properties until they are unlocked. Properties must be unlocked before they are destroyed.

The lock is automatically taken when setting individual properties, this function is only needed when you want to set several properties atomically or want to guarantee that properties being queried aren't freed in another thread.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_UnlockProperties

SDL_LockRWLockForReading

Lock the read/write lock for _read only_ operations.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
void SDL_LockRWLockForReading(SDL_RWLock *rwlock)
    SDL_ACQUIRE_SHARED(rwlock);
```

Function Parameters

rwlock	the read/write lock to lock

Remarks

This will block until the rwlock is available, which is to say it is not locked for writing by any other thread. Of all threads waiting to lock the rwlock, all may do so at the same time as long as they are requesting read-only access; if a thread wants to lock for writing, only one may do so at a time, and no other threads, read-only or not, may hold the lock at the same time.

It is legal for the owning thread to lock an already-locked rwlock for reading. It must unlock it the same number of times before it is actually made available for other threads in the system (this is known as a "recursive rwlock").

Note that locking for writing is not recursive (this is only available to read-only locks).

It is illegal to request a read-only lock from a thread that already holds the write lock. Doing so results in undefined behavior. Unlock the write lock before requesting a read-only lock. (But, of course, if you have the write lock, you don't need further locks to read in any case.)

This function does not fail; if rwlock is NULL, it will return immediately having locked nothing. If the rwlock is valid, this function will always block until it can lock the mutex, and return with it locked.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockRWLockForWriting

- SDL_TryLockRWLockForReading
- SDL_UnlockRWLock

SDL_LockRWLockForWriting

Lock the read/write lock for _write_ operations.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
void SDL_LockRWLockForWriting(SDL_RWLock *rwlock) SDL_ACQUIRE(rwlock);
```

Function Parameters

rwlock	the read/write lock to lock

Remarks

This will block until the rwlock is available, which is to say it is not locked for reading or writing by any other thread. Only one thread may hold the lock when it requests write access; all other threads, whether they also want to write or only want read-only access, must wait until the writer thread has released the lock.

It is illegal for the owning thread to lock an already-locked rwlock for writing (read-only may be locked recursively, writing can not). Doing so results in undefined behavior.

It is illegal to request a write lock from a thread that already holds a read-only lock. Doing so results in undefined behavior. Unlock the read-only lock before requesting a write lock.

This function does not fail; if rwlock is NULL, it will return immediately having locked nothing. If the rwlock is valid, this function will always block until it can lock the mutex, and return with it locked.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockRWLockForReading
- SDL_TryLockRWLockForWriting
- SDL_UnlockRWLock

SDL_LockSpinlock

Lock a spin lock by setting it to a non-zero value.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
void SDL_LockSpinlock(SDL_SpinLock *lock);
```

Function Parameters

lock	a pointer to a lock variable

Remarks

Please note that spinlocks are dangerous if you don't know what you're doing. Please be careful using any sort of spinlock!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_TryLockSpinlock
- SDL_UnlockSpinlock

SDL_LockSurface

Set up a surface for directly accessing the pixels.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_LockSurface(SDL_Surface *surface);
```

Function Parameters

surface	the SDL_Surface structure to be locked

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Between calls to SDL_LockSurface() / SDL_UnlockSurface(), you can write to and read from `surface->pixels`, using the pixel format stored in `surface->format`. Once you are done accessing the surface, you should use SDL_UnlockSurface() to release it.

Not all surfaces require locking. If `SDL_MUSTLOCK(surface)` evaluates to 0, then you can read and write to the surface at any time, and the pixel format of the surface will not change.

Code Examples

```
void modify_surface(SDL_Surface *surface) {

  /* Make the pixels pointer valid in the surface */

  SDL_LockSurface(surface);

  /* Surface is locked */
  /* Direct pixel access on surface here */

  SDL_UnlockSurface(surface);

  /* Surface is now unlocked */
```

```
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_MUSTLOCK
- SDL_UnlockSurface

SDL_LockTexture

Lock a portion of the texture for **write-only** pixel access.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_LockTexture(SDL_Texture *texture,
                    const SDL_Rect *rect,
                    void **pixels, int *pitch);
```

Function Parameters

texture	the texture to lock for access, which was created with `SDL_TEXTUREACCESS_STREAMING`
rect	an SDL_Rect structure representing the area to lock for access; NULL to lock the entire texture
pixels	this is filled in with a pointer to the locked pixels, appropriately offset by the locked area
pitch	this is filled in with the pitch of the locked pixels; the pitch is the length of one row in bytes

Return Value

Returns 0 on success or a negative error code if the texture is not valid or was not created with `SDL_TEXTUREACCESS_STREAMING`; call SDL_GetError() for more information.

Remarks

As an optimization, the pixels made available for editing don't necessarily contain the old texture data. This is a write-only operation, and if you need to keep a copy of the texture data you should do that at the application level.

You must use SDL_UnlockTexture() to unlock the pixels and apply any changes.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockTextureToSurface
- SDL_UnlockTexture

SDL_LockTextureToSurface

Lock a portion of the texture for **write-only** pixel access, and expose it as a SDL surface.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_LockTextureToSurface(SDL_Texture *texture,
                             const SDL_Rect *rect,
                             SDL_Surface **surface);
```

Function Parameters

texture	the texture to lock for access, which must be created with `SDL_TEXTUREACCESS_STREAMING`
rect	a pointer to the rectangle to lock for access. If the rect is NULL, the entire texture will be locked
surface	this is filled in with an SDL surface representing the locked area

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Besides providing an SDL_Surface instead of raw pixel data, this function operates like SDL_LockTexture.

As an optimization, the pixels made available for editing don't necessarily contain the old texture data. This is a write-only operation, and if you need to keep a copy of the texture data you should do that at the application level.

You must use SDL_UnlockTexture() to unlock the pixels and apply any changes.

The returned surface is freed internally after calling SDL_UnlockTexture() or SDL_DestroyTexture(). The caller should not free it.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockTexture
- SDL_UnlockTexture

SDL_log

Compute the natural logarithm of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_log(double x);
```

Function Parameters

x	floating point value. Must be greater than 0.

Return Value

Returns the natural logarithm of `x`

Remarks

Domain: `0 < x <= INF`

Range: `-INF <= y <= INF`

It is an error for `x` to be less than or equal to 0.

This function operates on double-precision floating point values, use SDL_logf for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_logf
- SDL_log10
- SDL_exp

SDL_Log

Log a message with SDL_LOG_CATEGORY_APPLICATION and SDL_LOG_PRIORITY_INFO.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_Log(SDL_PRINTF_FORMAT_STRING const char *fmt, ...)
    SDL_PRINTF_VARARG_FUNC(1);
```

Function Parameters

fmt	a printf() style message format string
...	additional parameters matching

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LogCritical
- SDL_LogDebug
- SDL_LogError
- SDL_LogInfo
- SDL_LogMessage
- SDL_LogMessageV
- SDL_LogVerbose
- SDL_LogWarn

SDL_log10

Compute the base-10 logarithm of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_log10(double x);
```

Function Parameters

x	floating point value. Must be greater than 0.

Return Value

Returns the logarithm of `x`

Remarks

Domain: `0 < x <= INF`

Range: `-INF <= y <= INF`

It is an error for `x` to be less than or equal to 0.

This function operates on double-precision floating point values, use SDL_log10f for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_log10f
- SDL_log
- SDL_pow

SDL_log10f

Compute the base-10 logarithm of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_log10f(float x);
```

Function Parameters

x	floating point value. Must be greater than 0.

Return Value

Returns the logarithm of `x`

Remarks

Domain: `0 < x <= INF`

Range: `-INF <= y <= INF`

It is an error for `x` to be less than or equal to 0.

This function operates on single-precision floating point values, use SDL_log10 for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_log10
- SDL_logf
- SDL_powf

SDL_LogCritical

Log a message with SDL_LOG_PRIORITY_CRITICAL.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogCritical(int category, SDL_PRINTF_FORMAT_STRING const char
    *fmt, ...) SDL_PRINTF_VARARG_FUNC(2);
```

Function Parameters

category	the category of the message
fmt	a printf() style message format string
...	additional parameters matching

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Log
- SDL_LogDebug
- SDL_LogError
- SDL_LogInfo
- SDL_LogMessage
- SDL_LogMessageV
- SDL_LogVerbose
- SDL_LogWarn

SDL_LogDebug

Log a message with SDL_LOG_PRIORITY_DEBUG.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogDebug(int category, SDL_PRINTF_FORMAT_STRING const char
    *fmt, ...) SDL_PRINTF_VARARG_FUNC(2);
```

Function Parameters

category	the category of the message
fmt	a printf() style message format string
...	additional parameters matching

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Log
- SDL_LogCritical
- SDL_LogError
- SDL_LogInfo
- SDL_LogMessage
- SDL_LogMessageV
- SDL_LogVerbose
- SDL_LogWarn

SDL_LogError

Log a message with SDL_LOG_PRIORITY_ERROR.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogError(int category, SDL_PRINTF_FORMAT_STRING const char
    *fmt, ...) SDL_PRINTF_VARARG_FUNC(2);
```

Function Parameters

category	the category of the message
fmt	a printf() style message format string
...	additional parameters matching

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Log
- SDL_LogCritical
- SDL_LogDebug
- SDL_LogInfo
- SDL_LogMessage
- SDL_LogMessageV
- SDL_LogVerbose
- SDL_LogWarn

SDL_logf

Compute the natural logarithm of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_logf(float x);
```

Function Parameters

x	floating point value. Must be greater than 0.

Return Value

Returns the natural logarithm of `x`

Remarks

Domain: `0 < x <= INF`

Range: `-INF <= y <= INF`

It is an error for `x` to be less than or equal to 0.

This function operates on single-precision floating point values, use SDL_log for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_log
- SDL_expf

SDL_LogGetPriority

Get the priority of a particular log category.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
SDL_LogPriority SDL_LogGetPriority(int category);
```

Function Parameters

category	the category to query

Return Value

Returns the SDL_LogPriority for the requested category

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LogSetPriority

SDL_LogInfo

Log a message with SDL_LOG_PRIORITY_INFO.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogInfo(int category, SDL_PRINTF_FORMAT_STRING const char *fmt,
    ...) SDL_PRINTF_VARARG_FUNC(2);
```

Function Parameters

category	the category of the message
fmt	a printf() style message format string
...	additional parameters matching

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Log
- SDL_LogCritical
- SDL_LogDebug
- SDL_LogError
- SDL_LogMessage
- SDL_LogMessageV
- SDL_LogVerbose
- SDL_LogWarn

SDL_LogMessage

Log a message with the specified category and priority.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogMessage(int category,
                    SDL_LogPriority priority,
                    SDL_PRINTF_FORMAT_STRING const char *fmt, ...)
                        SDL_PRINTF_VARARG_FUNC(3);
```

Function Parameters

category	the category of the message
priority	the priority of the message
fmt	a printf() style message format string
...	additional parameters matching

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Log
- SDL_LogCritical
- SDL_LogDebug
- SDL_LogError
- SDL_LogInfo
- SDL_LogMessageV
- SDL_LogVerbose
- SDL_LogWarn

SDL_LogMessageV

Log a message with the specified category and priority.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogMessageV(int category,
                     SDL_LogPriority priority,
                     SDL_PRINTF_FORMAT_STRING const char *fmt, va_list ap)
                         SDL_PRINTF_VARARG_FUNCV(3);
```

Function Parameters

category	the category of the message
priority	the priority of the message
fmt	a printf() style message format string
ap	a variable argument list

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Log
- SDL_LogCritical
- SDL_LogDebug
- SDL_LogError
- SDL_LogInfo
- SDL_LogMessage
- SDL_LogVerbose
- SDL_LogWarn

SDL_LogResetPriorities

Reset all priorities to default.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogResetPriorities(void);
```

Remarks

This is called by SDL_Quit().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LogSetAllPriority
- SDL_LogSetPriority

SDL_LogSetAllPriority

Set the priority of all log categories.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogSetAllPriority(SDL_LogPriority priority);
```

Function Parameters

priority	the SDL_LogPriority to assign

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LogResetPriorities
- SDL_LogSetPriority

SDL_LogSetPriority

Set the priority of a particular log category.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogSetPriority(int category,
                        SDL_LogPriority priority);
```

Function Parameters

category	the category to assign a priority to
priority	the SDL_LogPriority to assign

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LogGetPriority
- SDL_LogResetPriorities
- SDL_LogSetAllPriority

SDL_LogVerbose

Log a message with SDL_LOG_PRIORITY_VERBOSE.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogVerbose(int category, SDL_PRINTF_FORMAT_STRING const char
    *fmt, ...) SDL_PRINTF_VARARG_FUNC(2);
```

Function Parameters

category	the category of the message
fmt	a printf() style message format string
...	additional parameters matching

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Log
- SDL_LogCritical
- SDL_LogDebug
- SDL_LogError
- SDL_LogInfo
- SDL_LogMessage
- SDL_LogMessageV
- SDL_LogWarn

SDL_LogWarn

Log a message with SDL_LOG_PRIORITY_WARN.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_LogWarn(int category, SDL_PRINTF_FORMAT_STRING const char *fmt,
    ...) SDL_PRINTF_VARARG_FUNC(2);
```

Function Parameters

category	the category of the message
fmt	a printf() style message format string
...	additional parameters matching

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Log
- SDL_LogCritical
- SDL_LogDebug
- SDL_LogError
- SDL_LogInfo
- SDL_LogMessage
- SDL_LogMessageV
- SDL_LogVerbose

SDL_lround

Round `x` to the nearest integer representable as a long

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
long SDL_lround(double x);
```

Function Parameters

x	floating point value

Return Value

Returns the nearest integer to `x`

Remarks

Rounds `x` to the nearest integer. Values halfway between integers will be rounded away from zero.

Domain: `-INF <= x <= INF`

Range: `MIN_LONG <= y <= MAX_LONG`

This function operates on double-precision floating point values, use SDL_lround for single-precision floats. To get the result as a floating-point type, use SDL_round.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_lroundf
- SDL_round
- SDL_floor
- SDL_ceil
- SDL_trunc

SDL_lroundf

Round `x` to the nearest integer representable as a long

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
long SDL_lroundf(float x);
```

Function Parameters

x	floating point value

Return Value

Returns the nearest integer to `x`

Remarks

Rounds `x` to the nearest integer. Values halfway between integers will be rounded away from zero.

Domain: `-INF <= x <= INF`

Range: `MIN_LONG <= y <= MAX_LONG`

This function operates on single-precision floating point values, use SDL_lroundf for double-precision floats. To get the result as a floating-point type, use SDL_roundf,

Version

This function is available since SDL 3.0.0.

See Also

- SDL_lround
- SDL_roundf
- SDL_floorf
- SDL_ceilf
- SDL_truncf

SDL_main

An app-supplied function for program entry.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
int SDL_main(int argc, char *argv[]);
```

Remarks

Apps do not directly create this function; they should create a standard ANSI-C `main` function instead. If SDL needs to insert some startup code before `main` runs, or the platform doesn't actually _use_ a function called "main", SDL will do some macro magic to redefine `main` to `SDL_main` and provide its own `main`.

Apps should include `SDL_main.h` in the same file as their `main` function, and they should not use that symbol for anything else in that file, as it might get redefined.

This function is only provided by the app if it isn't using SDL_MAIN_USE_CALLBACKS.

Program startup is a surprisingly complex topic. Please see README/main-functions, (or docs/README-main-functions.md in the source tree) for a more detailed explanation.

Thread Safety

This is the program entry point.

Version

This function is available since SDL 3.0.0.

SDL_MapRGB

Map an RGB triple to an opaque pixel value for a given pixel format.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
Uint32 SDL_MapRGB(const SDL_PixelFormat * format,
                  Uint8 r, Uint8 g, Uint8 b);
```

Function Parameters

format	an SDL_PixelFormat structure describing the pixel format
r	the red component of the pixel in the range 0-255
g	the green component of the pixel in the range 0-255
b	the blue component of the pixel in the range 0-255

Return Value

Returns a pixel value

Remarks

This function maps the RGB color value to the specified pixel format and returns the pixel value best approximating the given RGB color value for the given pixel format.

If the format has a palette (8-bit) the index of the closest matching color in the palette will be returned.

If the specified pixel format has an alpha component it will be returned as all 1 bits (fully opaque).

If the pixel format bpp (color depth) is less than 32-bpp then the unused upper bits of the return value can safely be ignored (e.g., with a 16-bpp format the return value can be assigned to a Uint16, and similarly a Uint8 for an 8-bpp format).

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRGB
- SDL_GetRGBA
- SDL_MapRGBA

SDL_MapRGBA

Map an RGBA quadruple to a pixel value for a given pixel format.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
Uint32 SDL_MapRGBA(const SDL_PixelFormat * format,
                   Uint8 r, Uint8 g, Uint8 b,
                   Uint8 a);
```

Function Parameters

format	an SDL_PixelFormat structure describing the format of the pixel
r	the red component of the pixel in the range 0-255
g	the green component of the pixel in the range 0-255
b	the blue component of the pixel in the range 0-255
a	the alpha component of the pixel in the range 0-255

Return Value

Returns a pixel value

Remarks

This function maps the RGBA color value to the specified pixel format and returns the pixel value best approximating the given RGBA color value for the given pixel format.

If the specified pixel format has no alpha component the alpha value will be ignored (as it will be in formats with a palette).

If the format has a palette (8-bit) the index of the closest matching color in the palette will be returned.

If the pixel format bpp (color depth) is less than 32-bpp then the unused upper bits of the return value can safely be ignored (e.g., with a 16-bpp format the return value can be assigned to a Uint16, and similarly a Uint8 for an 8-bpp format).

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRGB
- SDL_GetRGBA
- SDL_MapRGB

SDL_MaximizeWindow

Request that the window be made as large as possible.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_MaximizeWindow(SDL_Window *window);
```

Function Parameters

window	the window to maximize

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Non-resizable windows can't be maximized. The window must have the SDL_WINDOW_RESIZABLE flag set, or this will have no effect.

On some windowing systems this request is asynchronous and the new window state may not have have been applied immediately upon the return of this function. If an immediate change is required, call SDL_SyncWindow() to block until the changes have taken effect.

When the window state changes, an SDL_EVENT_WINDOW_MAXIMIZED event will be emitted. Note that, as this is just a request, the windowing system can deny the state change.

When maximizing a window, whether the constraints set via SDL_SetWindowMaximumSize() are honored depends on the policy of the window manager. Win32 and macOS enforce the constraints when maximizing, while X11 and Wayland window managers may vary.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_MinimizeWindow
- SDL_RestoreWindow
- SDL_SyncWindow

SDL_MemoryBarrierAcquireFunction

Insert a memory acquire barrier.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
void SDL_MemoryBarrierAcquireFunction(void);
```

Remarks

Please refer to SDL_MemoryBarrierReleaseFunction for the details!

Thread Safety

Obviously this function is safe to use from any thread at any time, but if you find yourself needing this, you are probably dealing with some very sensitive code; be careful!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_MemoryBarrierReleaseFunction

SDL_MemoryBarrierReleaseFunction

Insert a memory release barrier.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
void SDL_MemoryBarrierReleaseFunction(void);
```

Remarks

Memory barriers are designed to prevent reads and writes from being reordered by the compiler and being seen out of order on multi-core CPUs.

A typical pattern would be for thread A to write some data and a flag, and for thread B to read the flag and get the data. In this case you would insert a release barrier between writing the data and the flag, guaranteeing that the data write completes no later than the flag is written, and you would insert an acquire barrier between reading the flag and reading the data, to ensure that all the reads associated with the flag have completed.

In this pattern you should always see a release barrier paired with an acquire barrier and you should gate the data reads/writes with a single flag variable.

For more information on these semantics, take a look at the blog post: http://preshing.com/20120913/acquire-and-release-semantics

Thread Safety

Obviously this macro is safe to use from any thread at any time, but if you find yourself needing this, you are probably dealing with some very sensitive code; be careful!

Version

This function is available since SDL 3.0.0.

SDL_Metal_CreateView

Create a CAMetalLayer-backed NSView/UIView and attach it to the specified window.

Header File

Defined in SDL3/SDL_metal.h

Syntax

```
SDL_MetalView SDL_Metal_CreateView(SDL_Window * window);
```

Function Parameters

window	the window

Return Value

Returns handle NSView or UIView

Remarks

On macOS, this does *not* associate a MTLDevice with the CAMetalLayer on its own. It is up to user code to do that.

The returned handle can be casted directly to a NSView or UIView. To access the backing CAMetalLayer, call SDL_Metal_GetLayer().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Metal_DestroyView
- SDL_Metal_GetLayer

SDL_Metal_DestroyView

Destroy an existing SDL_MetalView object.

Header File

Defined in SDL3/SDL_metal.h

Syntax

```
void SDL_Metal_DestroyView(SDL_MetalView view);
```

Function Parameters

view	the SDL_MetalView object

Remarks

This should be called before SDL_DestroyWindow, if SDL_Metal_CreateView was called after SDL_CreateWindow.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Metal_CreateView

SDL_Metal_GetLayer

Get a pointer to the backing CAMetalLayer for the given view.

Header File

Defined in SDL3/SDL_metal.h

Syntax

```
void* SDL_Metal_GetLayer(SDL_MetalView view);
```

Function Parameters

view	the SDL_MetalView object

Return Value

Returns a pointer

Version

This function is available since SDL 3.0.0.

SDL_MinimizeWindow

Request that the window be minimized to an iconic representation.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_MinimizeWindow(SDL_Window *window);
```

Function Parameters

window	the window to minimize

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

On some windowing systems this request is asynchronous and the new window state may not have have been applied immediately upon the return of this function. If an immediate change is required, call SDL_SyncWindow() to block until the changes have taken effect.

When the window state changes, an SDL_EVENT_WINDOW_MINIMIZED event will be emitted. Note that, as this is just a request, the windowing system can deny the state change.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_MaximizeWindow
- SDL_RestoreWindow
- SDL_SyncWindow

SDL_MixAudioFormat

Mix audio data in a specified format.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_MixAudioFormat(Uint8 * dst,
                       const Uint8 * src,
                       SDL_AudioFormat format,
                       Uint32 len, int volume);
```

Function Parameters

dst	the destination for the mixed audio
src	the source audio buffer to be mixed
format	the SDL_AudioFormat structure representing the desired audio format
len	the length of the audio buffer in bytes
volume	ranges from 0 - 128, and should be set to SDL_MIX_MAXVOLUME for full audio volume

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This takes an audio buffer `src` of `len` bytes of `format` data and mixes it into `dst`, performing addition, volume adjustment, and overflow clipping. The buffer pointed to by `dst` must also be `len` bytes of `format` data.

This is provided for convenience – you can mix your own audio data.

Do not use this function for mixing together more than two streams of sample data. The output from repeated application of this function may be distorted by clipping, because there is no accumulator with greater range than the input (not to mention this being an inefficient way of doing it).

It is a common misconception that this function is required to write audio data to an output stream in an audio callback. While you can do that, SDL_MixAudioFormat() is really only needed when you're mixing a single audio stream with a volume adjustment.

Code Examples

```
void MyAudioCallback(void *udata, Uint8 *stream, int len)
{
    extern SDL_AudioFormat deviceFormat;
    extern const Uint8 *mixData;
    SDL_memset(stream, 0, len); // make sure this is silence.
    // mix our audio against the silence, at 50% volume.
    SDL_MixAudioFormat(stream, mixData, deviceFormat, len,
        SDL_MIX_MAXVOLUME / 2);
}
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_modf

Split `x` into integer and fractional parts

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_modf(double x, double *y);
```

Function Parameters

x	floating point value
y	output pointer to store the integer part of `x`

Return Value

Returns the fractional part of `x`

Remarks

This function operates on double-precision floating point values, use SDL_modff for single-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_modff
- SDL_trunc
- SDL_fmod

SDL_modff

Split `x` into integer and fractional parts

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_modff(float x, float *y);
```

Function Parameters

x	floating point value
y	output pointer to store the integer part of `x`

Return Value

Returns the fractional part of `x`

Remarks

This function operates on single-precision floating point values, use SDL_modf for double-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_modf
- SDL_truncf
- SDL_fmodf

SDL_MostSignificantBitIndex32

Get the index of the most significant (set) bit in a 32-bit number.

Header File

Defined in SDL3/SDL_bits.h

Syntax

```
SDL_FORCE_INLINE int SDL_MostSignificantBitIndex32(Uint32 x);
```

Function Parameters

x	the 32-bit value to examine

Return Value

Returns the index of the most significant bit, or -1 if the value is 0.

Remarks

Result is undefined when called with 0. This operation can also be stated as "count leading zeroes" and "log base 2".

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_OnApplicationDidBecomeActive

Let iOS apps with external event handling report onApplicationDidBecomeActive.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_OnApplicationDidBecomeActive(void);
```

Remarks

This functions allows iOS apps that have their own event handling to hook into SDL to generate SDL events. This maps directly to an iOS-specific event, but since it doesn't do anything iOS-specific internally, it is available on all platforms, in case it might be useful for some specific paradigm. Most apps do not need to use this directly; SDL's internal event code will handle all this for windows created by SDL_CreateWindow!

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_OnApplicationDidChangeStatusBarOrientation

Let iOS apps with external event handling report onApplicationDidChangeStatusBarOrientation.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_OnApplicationDidChangeStatusBarOrientation(void);
```

Remarks

This functions allows iOS apps that have their own event handling to hook into SDL to generate SDL events. This maps directly to an iOS-specific event, but since it doesn't do anything iOS-specific internally, it is available on all platforms, in case it might be useful for some specific paradigm. Most apps do not need to use this directly; SDL's internal event code will handle all this for windows created by SDL_CreateWindow!

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_OnApplicationDidEnterBackground

Let iOS apps with external event handling report onApplicationDidEnterBackground.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_OnApplicationDidEnterBackground(void);
```

Remarks

This functions allows iOS apps that have their own event handling to hook into SDL to generate SDL events. This maps directly to an iOS-specific event, but since it doesn't do anything iOS-specific internally, it is available on all platforms, in case it might be useful for some specific paradigm. Most apps do not need to use this directly; SDL's internal event code will handle all this for windows created by SDL_CreateWindow!

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_OnApplicationDidReceiveMemoryWarning

Let iOS apps with external event handling report onApplicationDidReceiveMemoryWarning.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_OnApplicationDidReceiveMemoryWarning(void);
```

Remarks

This functions allows iOS apps that have their own event handling to hook into SDL to generate SDL events. This maps directly to an iOS-specific event, but since it doesn't do anything iOS-specific internally, it is available on all platforms, in case it might be useful for some specific paradigm. Most apps do not need to use this directly; SDL's internal event code will handle all this for windows created by SDL_CreateWindow!

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_OnApplicationWillEnterForeground

Let iOS apps with external event handling report onApplicationWillEnterForeground.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_OnApplicationWillEnterForeground(void);
```

Remarks

This functions allows iOS apps that have their own event handling to hook into SDL to generate SDL events. This maps directly to an iOS-specific event, but since it doesn't do anything iOS-specific internally, it is available on all platforms, in case it might be useful for some specific paradigm. Most apps do not need to use this directly; SDL's internal event code will handle all this for windows created by SDL_CreateWindow!

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_OnApplicationWillResignActive

Let iOS apps with external event handling report onApplicationWillResignActive.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_OnApplicationWillResignActive(void);
```

Remarks

This functions allows iOS apps that have their own event handling to hook into SDL to generate SDL events. This maps directly to an iOS-specific event, but since it doesn't do anything iOS-specific internally, it is available on all platforms, in case it might be useful for some specific paradigm. Most apps do not need to use this directly; SDL's internal event code will handle all this for windows created by SDL_CreateWindow!

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_OnApplicationWillTerminate

Let iOS apps with external event handling report onApplicationWillTerminate.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_OnApplicationWillTerminate(void);
```

Remarks

This functions allows iOS apps that have their own event handling to hook into SDL to generate SDL events. This maps directly to an iOS-specific event, but since it doesn't do anything iOS-specific internally, it is available on all platforms, in case it might be useful for some specific paradigm. Most apps do not need to use this directly; SDL's internal event code will handle all this for windows created by SDL_CreateWindow!

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_OpenAudioDevice

Open a specific audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
SDL_AudioDeviceID SDL_OpenAudioDevice(SDL_AudioDeviceID devid, const
    SDL_AudioSpec *spec);
```

Function Parameters

devid	the device instance id to open, or SDL_AUDIO_DEVICE_DEFAULT_OUTPUT or SDL_AUDIO_DEVICE_DEFAULT_CAPTURE for the most reasonable default device.
spec	the requested device configuration. Can be NULL to use reasonable defaults.

Return Value

Returns The device ID on success, 0 on error; call SDL_GetError() for more information.

Remarks

You can open both output and capture devices through this function. Output devices will take data from bound audio streams, mix it, and send it to the hardware. Capture devices will feed any bound audio streams with a copy of any incoming data.

An opened audio device starts out with no audio streams bound. To start audio playing, bind a stream and supply audio data to it. Unlike SDL2, there is no audio callback; you only bind audio streams and make sure they have data flowing into them (however, you can simulate SDL2's semantics fairly closely by using SDL_OpenAudioDeviceStream instead of this function).

If you don't care about opening a specific device, pass a `devid` of either `SDL_AUDIO_DEVICE_DEFAULT_OUTPUT` or `SDL_AUDIO_DEVICE_DEFAULT_CAPTURE`. In this case, SDL will try to pick the most reasonable default, and may also switch between physical devices seamlessly later, if the most reasonable default changes during the lifetime of this opened device (user changed the default in the OS's system preferences, the default got unplugged so the system jumped to a new

default, the user plugged in headphones on a mobile device, etc). Unless you have a good reason to choose a specific device, this is probably what you want.

You may request a specific format for the audio device, but there is no promise the device will honor that request for several reasons. As such, it's only meant to be a hint as to what data your app will provide. Audio streams will accept data in whatever format you specify and manage conversion for you as appropriate. SDL_GetAudioDeviceFormat can tell you the preferred format for the device before opening and the actual format the device is using after opening.

It's legal to open the same device ID more than once; each successful open will generate a new logical SDL_AudioDeviceID that is managed separately from others on the same physical device. This allows libraries to open a device separately from the main app and bind its own streams without conflicting.

It is also legal to open a device ID returned by a previous call to this function; doing so just creates another logical device on the same physical device. This may be useful for making logical groupings of audio streams.

This function returns the opened device ID on success. This is a new, unique SDL_AudioDeviceID that represents a logical device.

Some backends might offer arbitrary devices (for example, a networked audio protocol that can connect to an arbitrary server). For these, as a change from SDL2, you should open a default device ID and use an SDL hint to specify the target if you care, or otherwise let the backend figure out a reasonable default. Most backends don't offer anything like this, and often this would be an end user setting an environment variable for their custom need, and not something an application should specifically manage.

When done with an audio device, possibly at the end of the app's life, one should call SDL_CloseAudioDevice() on the returned device id.

Code Examples

```
SDL_AudioSpec want;
SDL_AudioDeviceID dev;

SDL_memset(&want, 0, sizeof(want)); /* or SDL_zero(want) */
want.format = SDL_AUDIO_F32;
want.channels = 2;
want.freq = 48000;

dev = SDL_OpenAudioDevice(SDL_AUDIO_DEVICE_DEFAULT_OUTPUT, &want);
if (dev == 0) {
    SDL_Log("Failed to open audio: %s", SDL_GetError());
} else {
    SDL_ResumeAudioDevice(dev); /* start audio playing. */
    SDL_Delay(5000); // let device play for 5 seconds
    SDL_CloseAudioDevice(dev);
}
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseAudioDevice
- SDL_GetAudioDeviceFormat

SDL_OpenAudioDeviceStream

Convenience function for straightforward audio init for the common case.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
SDL_AudioStream* SDL_OpenAudioDeviceStream(SDL_AudioDeviceID devid,
    const SDL_AudioSpec *spec, SDL_AudioStreamCallback callback, void
    *userdata);
```

Function Parameters

devid	an audio device to open, or SDL_AUDIO_DEVICE_DEFAULT_OUTPUT or SDL_AUDIO_DEVICE_DEFAULT_CAPTURE.
spec	the audio stream's data format. Required.
callback	A callback where the app will provide new data for playback, or receive new data for capture. Can be NULL, in which case the app will need to call SDL_PutAudioStreamData or SDL_GetAudioStreamData as necessary.
userdata	App-controlled pointer passed to callback. Can be NULL. Ignored if callback is NULL.

Return Value

Returns an audio stream on success, ready to use. NULL on error; call SDL_GetError() for more information. When done with this stream, call SDL_DestroyAudioStream to free resources and close the device.

Remarks

If all your app intends to do is provide a single source of PCM audio, this function allows you to do all your audio setup in a single call.

This is also intended to be a clean means to migrate apps from SDL2.

This function will open an audio device, create a stream and bind it. Unlike other methods of setup, the audio device will be closed when this stream is destroyed, so the app can treat the returned SDL_AudioStream as the only object needed to manage audio playback.

Also unlike other functions, the audio device begins paused. This is to map more closely to SDL2-style behavior, since there is no extra step here to bind

a stream to begin audio flowing. The audio device should be resumed with `SDL_ResumeAudioDevice(SDL_GetAudioStreamDevice(stream));`

This function works with both playback and capture devices.

The `spec` parameter represents the app's side of the audio stream. That is, for recording audio, this will be the output format, and for playing audio, this will be the input format.

If you don't care about opening a specific audio device, you can (and probably _should_), use SDL_AUDIO_DEVICE_DEFAULT_OUTPUT for playback and SDL_AUDIO_DEVICE_DEFAULT_CAPTURE for recording.

One can optionally provide a callback function; if NULL, the app is expected to queue audio data for playback (or unqueue audio data if capturing). Otherwise, the callback will begin to fire once the device is unpaused.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAudioStreamDevice
- SDL_ResumeAudioDevice

SDL_OpenCameraDevice

Open a video capture device (a "camera").

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
SDL_Camera* SDL_OpenCameraDevice(SDL_CameraDeviceID instance_id, const
    SDL_CameraSpec *spec);
```

Function Parameters

instance_id	the camera device instance ID
spec	The desired format for data the device will provide. Can be NULL.

Return Value

Returns device, or NULL on failure; call SDL_GetError() for more information.

Remarks

You can open the device with any reasonable spec, and if the hardware can't directly support it, it will convert data seamlessly to the requested format. This might incur overhead, including scaling of image data.

If you would rather accept whatever format the device offers, you can pass a NULL spec here and it will choose one for you (and you can use SDL_Surface's conversion/scaling functions directly if necessary).

You can call SDL_GetCameraFormat() to get the actual data format if passing a NULL spec here. You can see the exact specs a device can support without conversion with SDL_GetCameraSupportedSpecs().

SDL will not attempt to emulate framerate; it will try to set the hardware to the rate closest to the requested speed, but it won't attempt to limit or duplicate frames artificially; call SDL_GetCameraFormat() to see the actual framerate of the opened the device, and check your timestamps if this is crucial to your app!

Note that the camera is not usable until the user approves its use! On some platforms, the operating system will prompt the user to permit access to the camera, and they can choose Yes or No at that point. Until they do, the camera will not be usable. The app should either wait for an SDL_EVENT_CAMERA_DEVICE_APPROVED (or SDL_EVENT_CAMERA_DEVICE_DENIED) event, or poll SDL_IsCameraApproved() occasionally until it returns non-zero. On platforms that don't require explicit

user approval (and perhaps in places where the user previously permitted access), the approval event might come immediately, but it might come seconds, minutes, or hours later!

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetCameraDevices
- SDL_GetCameraFormat

SDL_OpenFileStorage

Opens up a container for local filesystem storage.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
SDL_Storage* SDL_OpenFileStorage(const char *path);
```

Function Parameters

path	the base path prepended to all storage paths, or NULL for no base path

Return Value

Returns a filesystem storage container on success or NULL on failure; call SDL_GetError() for more information.

Remarks

This is provided for development and tools. Portable applications should use SDL_OpenTitleStorage() for access to game data and SDL_OpenUserStorage() for access to user data.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseStorage
- SDL_GetStorageFileSize
- SDL_GetStorageSpaceRemaining
- SDL_OpenTitleStorage
- SDL_OpenUserStorage
- SDL_ReadStorageFile
- SDL_WriteStorageFile

SDL_OpenGamepad

Open a gamepad for use.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
SDL_Gamepad* SDL_OpenGamepad(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns a gamepad identifier or NULL if an error occurred; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseGamepad
- SDL_IsGamepad

SDL_OpenHaptic

Open a haptic device for use.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_Haptic* SDL_OpenHaptic(SDL_HapticID instance_id);
```

Function Parameters

instance_id	the haptic device instance ID

Return Value

Returns the device identifier or NULL on failure; call SDL_GetError() for more information.

Remarks

The index passed as an argument refers to the N'th haptic device on this system.

When opening a haptic device, its gain will be set to maximum and autocenter will be disabled. To modify these values use SDL_SetHapticGain() and SDL_SetHapticAutocenter().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseHaptic
- SDL_GetHaptics
- SDL_OpenHapticFromJoystick
- SDL_OpenHapticFromMouse
- SDL_SetHapticAutocenter
- SDL_SetHapticGain

SDL_OpenHapticFromJoystick

Open a haptic device for use from a joystick device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_Haptic* SDL_OpenHapticFromJoystick(SDL_Joystick *joystick);
```

Function Parameters

joystick	the SDL_Joystick to create a haptic device from

Return Value

Returns a valid haptic device identifier on success or NULL on failure; call SDL_GetError() for more information.

Remarks

You must still close the haptic device separately. It will not be closed with the joystick.

When opened from a joystick you should first close the haptic device before closing the joystick device. If not, on some implementations the haptic device will also get unallocated and you'll be unable to use force feedback on that device.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseHaptic
- SDL_IsJoystickHaptic

SDL_OpenHapticFromMouse

Try to open a haptic device from the current mouse.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
SDL_Haptic* SDL_OpenHapticFromMouse(void);
```

Return Value

Returns the haptic device identifier or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseHaptic
- SDL_IsMouseHaptic

SDL_OpenIO

Create a custom SDL_IOStream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_IOStream* SDL_OpenIO(const SDL_IOStreamInterface *iface, void
    *userdata);
```

Function Parameters

iface	The function pointers that implement this SDL_IOStream.
userdata	The app-controlled pointer that is passed to iface's functions when called.

Return Value

Returns a pointer to the allocated memory on success, or NULL on failure; call SDL_GetError() for more information.

Remarks

Applications do not need to use this function unless they are providing their own SDL_IOStream implementation. If you just need an SDL_IOStream to read/write a common data source, you should use the built-in implementations in SDL, like SDL_IOFromFile() or SDL_IOFromMem(), etc.

You must free the returned pointer with SDL_CloseIO().

This function makes a copy of `iface` and the caller does not need to keep this data around after this call.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseIO
- SDL_IOFromConstMem
- SDL_IOFromFile
- SDL_IOFromMem

SDL_OpenJoystick

Open a joystick for use.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
SDL_Joystick* SDL_OpenJoystick(SDL_JoystickID instance_id);
```

Function Parameters

instance_id	the joystick instance ID

Return Value

Returns a joystick identifier or NULL if an error occurred; call SDL_GetError() for more information.

Remarks

The joystick subsystem must be initialized before a joystick can be opened for use.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseJoystick

SDL_OpenSensor

Open a sensor for use.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
SDL_Sensor* SDL_OpenSensor(SDL_SensorID instance_id);
```

Function Parameters

instance_id	the sensor instance ID

Return Value

Returns an SDL_Sensor sensor object, or NULL if an error occurred.

Version

This function is available since SDL 3.0.0.

SDL_OpenStorage

Opens up a container using a client-provided storage interface.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
SDL_Storage* SDL_OpenStorage(const SDL_StorageInterface *iface, void
    *userdata);
```

Function Parameters

iface	the function table to be used by this container
userdata	the pointer that will be passed to the store interface

Return Value

Returns a storage container on success or NULL on failure; call SDL_GetError() for more information.

Remarks

Applications do not need to use this function unless they are providing their own SDL_Storage implementation. If you just need an SDL_Storage, you should use the built-in implementations in SDL, like SDL_OpenTitleStorage() or SDL_OpenUserStorage().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseStorage
- SDL_GetStorageFileSize
- SDL_GetStorageSpaceRemaining
- SDL_ReadStorageFile
- SDL_StorageReady
- SDL_WriteStorageFile

SDL_OpenTitleStorage

Opens up a read-only container for the application's filesystem.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
SDL_Storage* SDL_OpenTitleStorage(const char *override, SDL_PropertiesID
    props);
```

Function Parameters

override	a path to override the backend's default title root
props	a property list that may contain backend-specific information

Return Value

Returns a title storage container on success or NULL on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseStorage
- SDL_GetStorageFileSize
- SDL_OpenUserStorage
- SDL_ReadStorageFile

SDL_OpenURL

Open a URL/URI in the browser or other appropriate external application.

Header File

Defined in SDL3/SDL_misc.h

Syntax

```
int SDL_OpenURL(const char *url);
```

Function Parameters

url	A valid URL/URI to open. Use `file:///full/path/to/file` for local files, if supported.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Open a URL in a separate, system-provided application. How this works will vary wildly depending on the platform. This will likely launch what makes sense to handle a specific URL's protocol (a web browser for `http://`, etc), but it might also be able to launch file managers for directories and other things.

What happens when you open a URL varies wildly as well: your game window may lose focus (and may or may not lose focus if your game was fullscreen or grabbing input at the time). On mobile devices, your app will likely move to the background or your process might be paused. Any given platform may or may not handle a given URL.

If this is unimplemented (or simply unavailable) for a platform, this will fail with an error. A successful result does not mean the URL loaded, just that we launched _something_ to handle it (or at least believe we did).

All this to say: this function can be useful, but you should definitely test it on every platform you target.

Version

This function is available since SDL 3.0.0.

SDL_OpenUserStorage

Opens up a container for a user's unique read/write filesystem.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
SDL_Storage* SDL_OpenUserStorage(const char *org, const char *app,
    SDL_PropertiesID props);
```

Function Parameters

org	the name of your organization
app	the name of your application
props	a property list that may contain backend-specific information

Return Value

Returns a user storage container on success or NULL on failure; call SDL_GetError() for more information.

Remarks

While title storage can generally be kept open throughout runtime, user storage should only be opened when the client is ready to read/write files. This allows the backend to properly batch file operations and flush them when the container has been closed; ensuring safe and optimal save I/O.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CloseStorage
- SDL_GetStorageFileSize
- SDL_GetStorageSpaceRemaining
- SDL_OpenTitleStorage
- SDL_ReadStorageFile

- SDL_StorageReady
- SDL_WriteStorageFile

SDL_PauseAudioDevice

Use this function to pause audio playback on a specified device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_PauseAudioDevice(SDL_AudioDeviceID dev);
```

Function Parameters

dev	a device opened by SDL_OpenAudioDevice()

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function pauses audio processing for a given device. Any bound audio streams will not progress, and no audio will be generated. Pausing one device does not prevent other unpaused devices from running.

Unlike in SDL2, audio devices start in an _unpaused_ state, since an app has to bind a stream before any audio will flow. Pausing a paused device is a legal no-op.

Pausing a device can be useful to halt all audio without unbinding all the audio streams. This might be useful while a game is paused, or a level is loading, etc.

Physical devices can not be paused or unpaused, only logical devices created through SDL_OpenAudioDevice() can be.

Code Examples

```
extern SDL_AudioDeviceID devid;
SDL_PauseAudioDevice(devid); // audio callback is stopped when this
    returns.
SDL_Delay(5000); // audio device plays silence for 5 seconds
SDL_ResumeAudioDevice(devid); // audio callback starts running again.
```

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ResumeAudioDevice
- SDL_AudioDevicePaused

SDL_PauseHaptic

Pause a haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_PauseHaptic(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic device to pause

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Device must support the `SDL_HAPTIC_PAUSE` feature. Call SDL_ResumeHaptic() to resume playback.

Do not modify the effects nor add new ones while the device is paused. That can cause all sorts of weird errors.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ResumeHaptic

SDL_PeepEvents

Check the event queue for messages and optionally return them.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
int SDL_PeepEvents(SDL_Event *events, int numevents, SDL_EventAction
    action, Uint32 minType, Uint32 maxType);
```

Function Parameters

events	destination buffer for the retrieved events
numevents	if action is SDL_ADDEVENT, the number of events to add back to the event queue; if action is SDL_PEEKEVENT or SDL_GETEVENT, the maximum number of events to retrieve
action	action to take; see [[#action—Remarks]] for details
minType	minimum value of the event type to be considered; SDL_EVENT_FIRST is a safe choice
maxType	maximum value of the event type to be considered; SDL_EVENT_LAST is a safe choice

Return Value

Returns the number of events actually stored or a negative error code on failure; call SDL_GetError() for more information.

Remarks

`action` may be any of the following:

- `SDL_ADDEVENT`: up to `numevents` events will be added to the back of the event queue.
- `SDL_PEEKEVENT`: `numevents` events at the front of the event queue, within the specified minimum and maximum type, will be returned to the caller and will _not_ be removed from the queue.
- `SDL_GETEVENT`: up to `numevents` events at the front of the event queue, within the specified minimum and maximum type, will be returned to the caller and will be removed from the queue.

You may have to call SDL_PumpEvents() before calling this function. Otherwise, the events may not be ready to be filtered when you call SDL_PeepEvents().

This function is thread-safe.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PollEvent
- SDL_PumpEvents
- SDL_PushEvent

SDL_PenConnected

Checks whether a pen is still attached.

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
SDL_bool SDL_PenConnected(SDL_PenID instance_id);
```

Function Parameters

instance_id	A pen ID.

Return Value

Returns SDL_TRUE if "instance_id" is valid and the corresponding pen is attached, or SDL_FALSE otherwise.

Remarks

If a pen is detached, it will not show up for SDL_GetPens(). Other operations will still be available but may return default values.

Version

This function is available since SDL 3.0.0.

SDL_PlayHapticRumble

Run a simple rumble effect on a haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_PlayHapticRumble(SDL_Haptic *haptic, float strength, Uint32
    length);
```

Function Parameters

haptic	the haptic device to play the rumble effect on
strength	strength of the rumble to play as a 0-1 float value
length	length of the rumble to play in milliseconds

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_InitHapticRumble
- SDL_StopHapticRumble

SDL_PointInRect

Determine whether a point resides inside a rectangle.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_FORCE_INLINE SDL_bool SDL_PointInRect(const SDL_Point *p, const
    SDL_Rect *r);
```

Function Parameters

p	the point to test.
r	the rectangle to test.

Return Value

Returns SDL_TRUE if `p` is contained by `r`, SDL_FALSE otherwise.

Remarks

A point is considered part of a rectangle if both `p` and `r` are not NULL, and `p`'s x and y coordinates are ¿= to the rectangle's top left corner, and ¡ the rectangle's x+w and y+h. So a 1x1 rectangle considers point (0,0) as "inside" and (0,1) as not.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_PointInRectFloat

Determine whether a point resides inside a floating point rectangle.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_FORCE_INLINE SDL_bool SDL_PointInRectFloat(const SDL_FPoint *p,
    const SDL_FRect *r);
```

Function Parameters

p	the point to test.
r	the rectangle to test.

Return Value

Returns SDL_TRUE if `p` is contained by `r`, SDL_FALSE otherwise.

Remarks

A point is considered part of a rectangle if both `p` and `r` are not NULL, and `p`'s x and y coordinates are ¿= to the rectangle's top left corner, and ¡ the rectangle's x+w and y+h. So a 1x1 rectangle considers point (0,0) as "inside" and (0,1) as not.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_PollEvent

Poll for currently pending events.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
SDL_bool SDL_PollEvent(SDL_Event *event);
```

Function Parameters

event	the SDL_Event structure to be filled with the next event from the queue, or NULL

Return Value

Returns SDL_TRUE if this got an event or SDL_FALSE if there are none available.

Remarks

If `event` is not NULL, the next event is removed from the queue and stored in the SDL_Event structure pointed to by `event`. The 1 returned refers to this event, immediately stored in the SDL Event structure – not an event to follow.

If `event` is NULL, it simply returns 1 if there is an event in the queue, but will not remove it from the queue.

As this function may implicitly call SDL_PumpEvents(), you can only call this function in the thread that set the video mode.

SDL_PollEvent() is the favored way of receiving system events since it can be done from the main loop and does not suspend the main loop while waiting on an event to be posted.

The common practice is to fully process the event queue once every frame, usually as a first step before updating the game's state:

```
while (game_is_still_running) {
    SDL_Event event;
    while (SDL_PollEvent(&event)) { // poll until all events are handled!
        // decide what to do with this event.
    }

    // update game state, draw the current frame
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PushEvent
- SDL_WaitEvent
- SDL_WaitEventTimeout

SDL_PostSemaphore

Atomically increment a semaphore's value and wake waiting threads.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_PostSemaphore(SDL_Semaphore *sem);
```

Function Parameters

sem	the semaphore to increment

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_TryWaitSemaphore
- SDL_WaitSemaphore
- SDL_WaitSemaphoreTimeout

SDL_pow

Raise `x` to the power `y`

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_pow(double x, double y);
```

Function Parameters

x	the base
y	the exponent

Return Value

Returns `x` raised to the power `y`

Remarks

Domain: `-INF <= x <= INF`, `-INF <= y <= INF`

Range: `-INF <= z <= INF`

If `y` is the base of the natural logarithm (e), consider using SDL_exp instead.

This function operates on double-precision floating point values, use SDL_powf for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_powf
- SDL_exp
- SDL_log

SDL_powf

Raise `x` to the power `y`

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_powf(float x, float y);
```

Function Parameters

x	the base
y	the exponent

Return Value

Returns `x` raised to the power `y`

Remarks

Domain: `-INF <= x <= INF`, `-INF <= y <= INF`

Range: `-INF <= z <= INF`

If `y` is the base of the natural logarithm (e), consider using SDL_exp instead.

This function operates on single-precision floating point values, use SDL_powf for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_pow
- SDL_expf
- SDL_logf

SDL_PremultiplyAlpha

Premultiply the alpha on a block of pixels.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_PremultiplyAlpha(int width, int height, SDL_PixelFormatEnum
    src_format, const void *src, int src_pitch, SDL_PixelFormatEnum
    dst_format, void *dst, int dst_pitch);
```

Function Parameters

width	the width of the block to convert, in pixels
height	the height of the block to convert, in pixels
src_format	an SDL_PixelFormatEnum value of the `src` pixels format
src	a pointer to the source pixels
src_pitch	the pitch of the source pixels, in bytes
dst_format	an SDL_PixelFormatEnum value of the `dst` pixels format
dst	a pointer to be filled in with premultiplied pixel data
dst_pitch	the pitch of the destination pixels, in bytes

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This is safe to use with src == dst, but not for other overlapping areas.

This function is currently only implemented for SDL_PIXELFORMAT_ARGB8888.

Version

This function is available since SDL 3.0.0.

SDL_PumpEvents

Pump the event loop, gathering events from the input devices.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
void SDL_PumpEvents(void);
```

Remarks

This function updates the event queue and internal input device state. **WARNING**: This should only be run in the thread that initialized the video subsystem, and for extra safety, you should consider only doing those things on the main thread in any case.

SDL_PumpEvents() gathers all the pending input information from devices and places it in the event queue. Without calls to SDL_PumpEvents() no events would ever be placed on the queue. Often the need for calls to SDL_PumpEvents() is hidden from the user since SDL_PollEvent() and SDL_WaitEvent() implicitly call SDL_PumpEvents(). However, if you are not polling or waiting for events (e.g. you are filtering them), then you must call SDL_PumpEvents() to force an event queue update.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PollEvent
- SDL_WaitEvent

SDL_PushEvent

Add an event to the event queue.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
int SDL_PushEvent(SDL_Event *event);
```

Function Parameters

event	the SDL_Event to be added to the queue

Return Value

Returns 1 on success, 0 if the event was filtered, or a negative error code on failure; call SDL_GetError() for more information. A common reason for error is the event queue being full.

Remarks

The event queue can actually be used as a two way communication channel. Not only can events be read from the queue, but the user can also push their own events onto it. `event` is a pointer to the event structure you wish to push onto the queue. The event is copied into the queue, and the caller may dispose of the memory pointed to after SDL_PushEvent() returns.

Note: Pushing device input events onto the queue doesn't modify the state of the device within SDL.

This function is thread-safe, and can be called from other threads safely.

Note: Events pushed onto the queue with SDL_PushEvent() get passed through the event filter but events added with SDL_PeepEvents() do not.

For pushing application-specific events, please use SDL_RegisterEvents() to get an event type that does not conflict with other code that also wants its own custom event types.

Code Examples

```
// Default Usage:
SDL_Event sdlevent;
sdlevent.type = SDL_EVENT_KEY_DOWN;
sdlevent.key.keysym.sym = SDLK_1;
```

```
SDL_PushEvent(&sdlevent);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PeepEvents
- SDL_PollEvent
- SDL_RegisterEvents

SDL_PutAudioStreamData

Add data to the stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_PutAudioStreamData(SDL_AudioStream *stream, const void *buf, int
    len);
```

Function Parameters

stream	The stream the audio data is being added to
buf	A pointer to the audio data to add
len	The number of bytes to write to the stream

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This data must match the format/channels/samplerate specified in the latest call to SDL_SetAudioStreamFormat, or the format specified when creating the stream if it hasn't been changed.

Note that this call simply copies the unconverted data for later. This is different than SDL2, where data was converted during the Put call and the Get call would just dequeue the previously-converted data.

Thread Safety

It is safe to call this function from any thread, but if the stream has a callback set, the caller might need to manage extra locking.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ClearAudioStream
- SDL_FlushAudioStream
- SDL_GetAudioStreamData
- SDL_GetAudioStreamQueued

SDL_QueryTexture

Query the attributes of a texture.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_QueryTexture(SDL_Texture *texture, SDL_PixelFormatEnum *format,
    int *access, int *w, int *h);
```

Function Parameters

texture	the texture to query
format	a pointer filled in with the raw format of the texture; the actual format may differ, but pixel transfers will use this format (one of the SDL_PixelFormatEnum values). This argument can be NULL if you don't need this information.
access	a pointer filled in with the actual access to the texture (one of the SDL_TextureAccess values). This argument can be NULL if you don't need this information.
w	a pointer filled in with the width of the texture in pixels. This argument can be NULL if you don't need this information.
h	a pointer filled in with the height of the texture in pixels. This argument can be NULL if you don't need this information.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Code Examples

```
SDL_Texture* source;

// loading etc ...

int w, h;
SDL_QueryTexture(source, NULL, NULL, &w, &h);
```

Version

This function is available since SDL 3.0.0.

SDL_Quit

Clean up all initialized subsystems.

Header File

Defined in SDL3/SDL_init.h

Syntax

```
void SDL_Quit(void);
```

Remarks

You should call this function even if you have already shutdown each initialized subsystem with SDL_QuitSubSystem(). It is safe to call this function even in the case of errors in initialization.

You can use this function with atexit() to ensure that it is run when your application is shutdown, but it is not wise to do this from a library or other dynamically loaded code.

Code Examples

```
#include <SDL3/SDL.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    if (SDL_Init(SDL_INIT_EVENTS) != 0) {
        fprintf(stderr, "Unable to initialize SDL: %s\n",
            SDL_GetError());
        return 1;
    }

    /* ... */

    SDL_Quit();

    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Init
- SDL_QuitSubSystem

SDL_QuitSubSystem

Shut down specific SDL subsystems.

Header File

Defined in SDL3/SDL_init.h

Syntax

```
void SDL_QuitSubSystem(Uint32 flags);
```

Function Parameters

flags	any of the flags used by SDL_Init(); see SDL_Init for details.

Remarks

You still need to call SDL_Quit() even if you close all open subsystems with SDL_QuitSubSystem().

Code Examples

```
#include <SDL3/SDL.h>

extern void display_graph(void);

/* ... */

int main(int argc, char **argv) {
    int sdl_initialized = 0;
    sdl_initialized = !SDL_Init(0);

    /* ... console stuff ... */

    if (sdl_initialized && SDL_InitSubSystem(SDL_INIT_VIDEO)) {
        display_graph();
        SDL_QuitSubSystem(SDL_INIT_VIDEO);
    }

    /* ... more console stuff ... */

    if (sdl_initialized) {
        SDL_Quit();
    }
    return 0;
```

```
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_InitSubSystem
- SDL_Quit

SDL_RaiseWindow

Raise a window above other windows and set the input focus.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_RaiseWindow(SDL_Window *window);
```

Function Parameters

window	the window to raise

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_ReadIO

Read from a data source.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
size_t SDL_ReadIO(SDL_IOStream *context, void *ptr, size_t size);
```

Function Parameters

context	a pointer to an SDL_IOStream structure
ptr	a pointer to a buffer to read data into
size	the number of bytes to read from the data source.

Return Value

Returns the number of bytes read, or 0 on end of file or other error.

Remarks

This function reads up `size` bytes from the data source to the area pointed at by `ptr`. This function may read less bytes than requested. It will return zero when the data stream is completely read, or on error. To determine if there was an error or all data was read, call SDL_GetIOStatus().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_WriteIO
- SDL_GetIOStatus

SDL_ReadS16BE

Use this function to read 16 bits of big-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadS16BE(SDL_IOStream *src, Sint16 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadS16LE

Use this function to read 16 bits of little-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadS16LE(SDL_IOStream *src, Sint16 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadS32BE

Use this function to read 32 bits of big-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadS32BE(SDL_IOStream *src, Sint32 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadS32LE

Use this function to read 32 bits of little-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadS32LE(SDL_IOStream *src, Sint32 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadS64BE

Use this function to read 64 bits of big-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadS64BE(SDL_IOStream *src, Sint64 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadS64LE

Use this function to read 64 bits of little-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadS64LE(SDL_IOStream *src, Sint64 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadStorageFile

Synchronously read a file from a storage container into a client-provided buffer.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
int SDL_ReadStorageFile(SDL_Storage *storage, const char *path, void
    *destination, Uint64 length);
```

Function Parameters

storage	a storage container to read from
path	the relative path of the file to read
destination	a client-provided buffer to read the file into
length	the length of the destination buffer

Return Value

Returns 0 if the file was read, a negative value otherwise; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetStorageFileSize
- SDL_StorageReady
- SDL_WriteStorageFile

SDL_ReadSurfacePixel

Retrieves a single pixel from a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_ReadSurfacePixel(SDL_Surface *surface, int x, int y, Uint8 *r,
    Uint8 *g, Uint8 *b, Uint8 *a);
```

Function Parameters

surface	the surface to read
x	the horizontal coordinate, 0 ¡= x ¡ width
y	the vertical coordinate, 0 ¡= y ¡ height
r	a pointer filled in with the red channel, 0-255, or NULL to ignore this channel
g	a pointer filled in with the green channel, 0-255, or NULL to ignore this channel
b	a pointer filled in with the blue channel, 0-255, or NULL to ignore this channel
a	a pointer filled in with the alpha channel, 0-255, or NULL to ignore this channel

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function prioritizes correctness over speed: it is suitable for unit tests, but is not intended for use in a game engine.

Like SDL_GetRGBA, this uses the entire 0..255 range when converting color components from pixel formats with less than 8 bits per RGB component.

Version

This function is available since SDL 3.0.0.

SDL_ReadU16BE

Use this function to read 16 bits of big-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadU16BE(SDL_IOStream *src, Uint16 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadU16LE

Use this function to read 16 bits of little-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadU16LE(SDL_IOStream *src, Uint16 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadU32BE

Use this function to read 32 bits of big-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadU32BE(SDL_IOStream *src, Uint32 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadU32LE

Use this function to read 32 bits of little-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadU32LE(SDL_IOStream *src, Uint32 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadU64BE

Use this function to read 64 bits of big-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadU64BE(SDL_IOStream *src, Uint64 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadU64LE

Use this function to read 64 bits of little-endian data from an SDL_IOStream and return in native format.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadU64LE(SDL_IOStream *src, Uint64 *value);
```

Function Parameters

src	the stream from which to read data
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the data returned will be in the native byte order.

Version

This function is available since SDL 3.0.0.

SDL_ReadU8

Use this function to read a byte from an SDL_IOStream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_ReadU8(SDL_IOStream *src, Uint8 *value);
```

Function Parameters

src	the SDL_IOStream to read from
value	a pointer filled in with the data read

Return Value

Returns SDL_TRUE on success or SDL_FALSE on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_RectEmpty

Determine whether a rectangle has no area.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_FORCE_INLINE SDL_bool SDL_RectEmpty(const SDL_Rect *r);
```

Function Parameters

r	the rectangle to test.

Return Value

Returns SDL_TRUE if the rectangle is "empty", SDL_FALSE otherwise.

Remarks

A rectangle is considered "empty" for this function if `r` is NULL, or if `r`'s width and/or height are ¡= 0.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_RectEmptyFloat

Determine whether a floating point rectangle has no area.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_FORCE_INLINE SDL_bool SDL_RectEmptyFloat(const SDL_FRect *r);
```

Function Parameters

r	the rectangle to test.

Return Value

Returns SDL_TRUE if the rectangle is "empty", SDL_FALSE otherwise.

Remarks

A rectangle is considered "empty" for this function if `r` is NULL, or if `r`'s width and/or height are ¡= 0.0f.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_RectsEqual

Determine whether two rectangles are equal.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_FORCE_INLINE SDL_bool SDL_RectsEqual(const SDL_Rect *a, const
    SDL_Rect *b);
```

Function Parameters

a	the first rectangle to test.
b	the second rectangle to test.

Return Value

Returns SDL_TRUE if the rectangles are equal, SDL_FALSE otherwise.

Remarks

Rectangles are considered equal if both are not NULL and each of their x, y, width and height match.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_RectsEqualEpsilon

Determine whether two floating point rectangles are equal, within some given epsilon.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_FORCE_INLINE SDL_bool SDL_RectsEqualEpsilon(const SDL_FRect *a,
    const SDL_FRect *b, const float epsilon);
```

Function Parameters

a	the first rectangle to test.
b	the second rectangle to test.

Return Value

Returns SDL_TRUE if the rectangles are equal, SDL_FALSE otherwise.

Remarks

Rectangles are considered equal if both are not NULL and each of their x, y, width and height are within `epsilon` of each other. If you don't know what value to use for `epsilon`, you should call the SDL_RectsEqualFloat function instead.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RectsEqualFloat

SDL_RectsEqualFloat

Determine whether two floating point rectangles are equal, within a default epsilon.

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
SDL_FORCE_INLINE SDL_bool SDL_RectsEqualFloat(const SDL_FRect *a, const
    SDL_FRect *b);
```

Function Parameters

a	the first rectangle to test.
b	the second rectangle to test.

Return Value

Returns SDL_TRUE if the rectangles are equal, SDL_FALSE otherwise.

Remarks

Rectangles are considered equal if both are not NULL and each of their x, y, width and height are within SDL_FLT_EPSILON of each other. This is often a reasonable way to compare two floating point rectangles and deal with the slight precision variations in floating point calculations that tend to pop up.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RectsEqualEpsilon

SDL_RegisterApp

Register a win32 window class for SDL's use.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
int SDL_RegisterApp(const char *name, Uint32 style, void *hInst);
```

Function Parameters

name	the window class name, in UTF-8 encoding. If NULL, SDL currently uses "SDL_app" but this isn't guaranteed.
style	the value to use in WNDCLASSEX::style. If `name` is NULL, SDL currently uses `(CS_BYTEALIGNCLIENT | CS_OWNDC)` regardless of what is specified here.
hInst	the HINSTANCE to use in WNDCLASSEX::hInstance. If zero, SDL will use `GetModuleHandle(NULL)` instead.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This can be called to set the application window class at startup. It is safe to call this multiple times, as long as every call is eventually paired with a call to SDL_UnregisterApp, but a second registration attempt while a previous registration is still active will be ignored, other than to increment a counter.

Most applications do not need to, and should not, call this directly; SDL will call it when initializing the video subsystem.

Version

This function is available since SDL 3.0.0.

SDL_RegisterEvents

Allocate a set of user-defined events, and return the beginning event number for that set of events.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
Uint32 SDL_RegisterEvents(int numevents);
```

Function Parameters

numevents	the number of events to be allocated

Return Value

Returns the beginning event number, or 0 if numevents is invalid or if there are not enough user-defined events left.

Code Examples

¡¡Include(SDL_UserEvent, , , from="== Code Examples ==", to="== Remarks ==")¿¿

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PushEvent

SDL_ReleaseCameraFrame

Release a frame of video acquired from a camera.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
int SDL_ReleaseCameraFrame(SDL_Camera *camera, SDL_Surface *frame);
```

Function Parameters

camera	opened camera device
frame	The video frame surface to release.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Let the back-end re-use the internal buffer for camera.

This function _must_ be called only on surface objects returned by SDL_AcquireCameraFrame(). This function should be called as quickly as possible after acquisition, as SDL keeps a small FIFO queue of surfaces for video frames; if surfaces aren't released in a timely manner, SDL may drop upcoming video frames from the camera.

If the app needs to keep the surface for a significant time, they should make a copy of it and release the original.

The app should not use the surface again after calling this function; assume the surface is freed and the pointer is invalid.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AcquireCameraFrame

SDL_ReloadGamepadMappings

Reinitialize the SDL mapping database to its initial state.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_ReloadGamepadMappings(void);
```

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This will generate gamepad events as needed if device mappings change.

Version

This function is available since SDL 3.0.0.

SDL_RemovePath

Remove a file or an empty directory.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
int SDL_RemovePath(const char *path);
```

Function Parameters

path	the path of the directory to enumerate

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_RemoveStoragePath

Remove a file or an empty directory in a writable storage container.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
int SDL_RemoveStoragePath(SDL_Storage *storage, const char *path);
```

Function Parameters

storage	a storage container
path	the path of the directory to enumerate

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StorageReady

SDL_RemoveTimer

Remove a timer created with SDL_AddTimer().

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
SDL_bool SDL_RemoveTimer(SDL_TimerID id);
```

Function Parameters

id	the ID of the timer to remove

Return Value

Returns SDL_TRUE if the timer is removed or SDL_FALSE if the timer wasn't found.

Code Examples

```
extern void *my_callback_param;
extern Uint32 SDLCALL my_callbackfunc(Uint32 interval, void *param);

Uint32 delay = (33 / 10) * 10; /* To round it down to the nearest 10 ms
    */

/* ... */

SDL_TimerID my_timer_id = SDL_AddTimer(delay, my_callbackfunc,
    my_callback_param);

/* ... */

SDL_RemoveTimer(my_timer_id);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AddTimer

SDL_RenamePath

Rename a file or directory.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
int SDL_RenamePath(const char *oldpath, const char *newpath);
```

Function Parameters

oldpath	the old path
newpath	the new path

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_RenameStoragePath

Rename a file or directory in a writable storage container.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
int SDL_RenameStoragePath(SDL_Storage *storage, const char *oldpath,
    const char *newpath);
```

Function Parameters

storage	a storage container
oldpath	the old path
newpath	the new path

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StorageReady

SDL_RenderClear

Clear the current rendering target with the drawing color.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderClear(SDL_Renderer *renderer);
```

Function Parameters

renderer	the rendering context

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function clears the entire rendering target, ignoring the viewport and the clip rectangle.

Code Examples

```
#include <SDL3/SDL.h>

int main(int argc, char* argv[])
{
        SDL_Window* window;
        SDL_Renderer* renderer;

        /* Initialize SDL. */
        if (SDL_Init(SDL_INIT_VIDEO) < 0)
                return 1;

        /* Create the window where we will draw. */
        window = SDL_CreateWindow("SDL_RenderClear",
                        512, 512,
                        0);

        /* We must call SDL_CreateRenderer in order for draw calls to
            affect this window. */
```

```
        renderer = SDL_CreateRenderer(window, NULL, 0);

        /* Select the color for drawing. It is set to red here. */
        SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);

        /* Clear the entire screen to our selected color. */
        SDL_RenderClear(renderer);

        /* Up until now everything was drawn behind the scenes.
           This will show the new, red contents of the window. */
        SDL_RenderPresent(renderer);

        /* Give us time to see the window. */
        SDL_Delay(5000);

        /* Always be sure to clean up */
        SDL_Quit();
        return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderDrawColor

SDL_RenderClipEnabled

Get whether clipping is enabled on the given renderer.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_bool SDL_RenderClipEnabled(SDL_Renderer *renderer);
```

Function Parameters

renderer	the rendering context

Return Value

Returns SDL_TRUE if clipping is enabled or SDL_FALSE if not; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderClipRect
- SDL_SetRenderClipRect

SDL_RenderCoordinatesFromWindow

Get a point in render coordinates when given a point in window coordinates.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderCoordinatesFromWindow(SDL_Renderer *renderer, float
    window_x, float window_y, float *x, float *y);
```

Function Parameters

renderer	the rendering context
window_x	the x coordinate in window coordinates
window_y	the y coordinate in window coordinates
x	a pointer filled with the x coordinate in render coordinates
y	a pointer filled with the y coordinate in render coordinates

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderLogicalPresentation
- SDL_SetRenderScale

SDL_RenderCoordinatesToWindow

Get a point in window coordinates when given a point in render coordinates.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderCoordinatesToWindow(SDL_Renderer *renderer, float x, float
    y, float *window_x, float *window_y);
```

Function Parameters

renderer	the rendering context
x	the x coordinate in render coordinates
y	the y coordinate in render coordinates
window_x	a pointer filled with the x coordinate in window coordinates
window_y	a pointer filled with the y coordinate in window coordinates

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetRenderLogicalPresentation
- SDL_SetRenderScale

SDL_RenderFillRect

Fill a rectangle on the current rendering target with the drawing color at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderFillRect(SDL_Renderer *renderer, const SDL_FRect *rect);
```

Function Parameters

renderer	The renderer which should fill a rectangle.
rect	A pointer to the destination rectangle, or NULL for the entire rendering target.

Return Value

Returns 0 on success, or -1 on error

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderFillRects

SDL_RenderFillRects

Fill some number of rectangles on the current rendering target with the drawing color at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderFillRects(SDL_Renderer *renderer, const SDL_FRect *rects,
    int count);
```

Function Parameters

renderer	The renderer which should fill multiple rectangles.
rects	A pointer to an array of destination rectangles.
count	The number of rectangles.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderFillRect

SDL_RenderGeometry

Render a list of triangles, optionally using a texture and indices into the vertex array Color and alpha modulation is done per vertex (SDL_SetTextureColorMod and SDL_SetTextureAlphaMod are ignored).

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderGeometry(SDL_Renderer *renderer,
                       SDL_Texture *texture,
                       const SDL_Vertex *vertices, int num_vertices,
                       const int *indices, int num_indices);
```

Function Parameters

renderer	The rendering context.
texture	(optional) The SDL texture to use.
vertices	Vertices.
num_vertices	Number of vertices.
indices	(optional) An array of integer indices into the 'vertices' array, if NULL all vertices will be rendered in sequential order.
num_indices	Number of indices.

Return Value

Returns 0 on success, or -1 if the operation is not supported

Code Examples

```
#include <SDL3/SDL.h>
#include <SDL3/SDL_main.h>

int main(int argc, char *argv[])
{
  SDL_bool quit = SDL_FALSE;
  SDL_Window *window = SDL_CreateWindow("Triangle Example", 800, 600, 0);

  SDL_Renderer *renderer = SDL_CreateRenderer(window, NULL,
      SDL_RENDERER_PRESENTVSYNC);
```

```
  #define vertLen 3
  SDL_Vertex vert[vertLen];

  // center
  vert[0].position.x = 400;
  vert[0].position.y = 150;
  vert[0].color.r = 1.0;
  vert[0].color.g = 0.0;
  vert[0].color.b = 0.0;
  vert[0].color.a = 1.0;

  // left
  vert[1].position.x = 200;
  vert[1].position.y = 450;
  vert[1].color.r = 0.0;
  vert[1].color.g = 0.0;
  vert[1].color.b = 1.0;
  vert[1].color.a = 1.0;

  // right
  vert[2].position.x = 600;
  vert[2].position.y = 450;
  vert[2].color.r = 0.0;
  vert[2].color.g = 1.0;
  vert[2].color.b = 0.0;
  vert[2].color.a = 1.0;

  while (!quit) {
   SDL_Event ev;
   while (SDL_PollEvent(&ev) != 0) {
      switch(ev.type) {
        case SDL_EVENT_QUIT:
        quit = SDL_TRUE;
        break;
      }
    }
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
    SDL_RenderClear(renderer);

    SDL_RenderGeometry(renderer, NULL, vert, vertLen, NULL, 0);

    SDL_RenderPresent(renderer);
  }

  SDL_DestroyRenderer(renderer);
  SDL_DestroyWindow(window);
  SDL_Quit();
  return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderGeometryRaw

SDL_RenderGeometryRaw

Render a list of triangles, optionally using a texture and indices into the vertex arrays Color and alpha modulation is done per vertex (SDL_SetTextureColorMod and SDL_SetTextureAlphaMod are ignored).

Header File

Defined in SDL3/SDL_render.h

Syntax

```
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);
```

Function Parameters

renderer	The rendering context.
texture	(optional) The SDL texture to use.
xy	Vertex positions
xy_stride	Byte size to move from one element to the next element
color	Vertex colors (as SDL_Color)
color_stride	Byte size to move from one element to the next element
uv	Vertex normalized texture coordinates
uv_stride	Byte size to move from one element to the next element
num_vertices	Number of vertices.
indices	(optional) An array of indices into the 'vertices' arrays, if NULL all vertices will be rendered in sequential order.
num_indices	Number of indices.
size_indices	Index size: 1 (byte), 2 (short), 4 (int)

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderGeometry

SDL_RenderGeometryRawFloat

Render a list of triangles, optionally using a texture and indices into the vertex arrays Color and alpha modulation is done per vertex (SDL_SetTextureColorMod and SDL_SetTextureAlphaMod are ignored).

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderGeometryRawFloat(SDL_Renderer *renderer,
                     SDL_Texture *texture,
                     const float *xy, int xy_stride,
                     const SDL_FColor *color, int color_stride,
                     const float *uv, int uv_stride,
                     int num_vertices,
                     const void *indices, int num_indices, int
                         size_indices);
```

Function Parameters

renderer	The rendering context.
texture	(optional) The SDL texture to use.
xy	Vertex positions
xy_stride	Byte size to move from one element to the next element
color	Vertex colors (as SDL_FColor)
color_stride	Byte size to move from one element to the next element
uv	Vertex normalized texture coordinates
uv_stride	Byte size to move from one element to the next element
num_vertices	Number of vertices.
indices	(optional) An array of indices into the 'vertices' arrays, if NULL all vertices will be rendered in sequential order.
num_indices	Number of indices.
size_indices	Index size: 1 (byte), 2 (short), 4 (int)

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderGeometry
- SDL_RenderGeometryRaw

SDL_RenderLine

Draw a line on the current rendering target at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderLine(SDL_Renderer *renderer, float x1, float y1, float x2,
    float y2);
```

Function Parameters

renderer	The renderer which should draw a line.
x1	The x coordinate of the start point.
y1	The y coordinate of the start point.
x2	The x coordinate of the end point.
y2	The y coordinate of the end point.

Return Value

Returns 0 on success, or -1 on error

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderLines

SDL_RenderLines

Draw a series of connected lines on the current rendering target at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderLines(SDL_Renderer *renderer, const SDL_FPoint *points,
    int count);
```

Function Parameters

renderer	The renderer which should draw multiple lines.
points	The points along the lines
count	The number of points, drawing count-1 lines

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderLine

SDL_RenderPoint

Draw a point on the current rendering target at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderPoint(SDL_Renderer *renderer, float x, float y);
```

Function Parameters

renderer	The renderer which should draw a point.
x	The x coordinate of the point.
y	The y coordinate of the point.

Return Value

Returns 0 on success, or -1 on error

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderPoints

SDL_RenderPoints

Draw multiple points on the current rendering target at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderPoints(SDL_Renderer *renderer, const SDL_FPoint *points,
    int count);
```

Function Parameters

renderer	The renderer which should draw multiple points.
points	The points to draw
count	The number of points to draw

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderPoint

SDL_RenderPresent

Update the screen with any rendering performed since the previous call.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderPresent(SDL_Renderer *renderer);
```

Function Parameters

renderer	the rendering context

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

SDL's rendering functions operate on a backbuffer; that is, calling a rendering function such as SDL_RenderLine() does not directly put a line on the screen, but rather updates the backbuffer. As such, you compose your entire scene and *present* the composed backbuffer to the screen as a complete picture.

Therefore, when using SDL's rendering API, one does all drawing intended for the frame, and then calls this function once per frame to present the final drawing to the user.

The backbuffer should be considered invalidated after each present; do not assume that previous contents will exist between frames. You are strongly encouraged to call SDL_RenderClear() to initialize the backbuffer before starting each new frame's drawing, even if you plan to overwrite every pixel.

Code Examples

Please refer to the code example in SDL_RenderClear.

Thread Safety

You may only call this function on the main thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderClear
- SDL_RenderLine
- SDL_RenderLines
- SDL_RenderPoint
- SDL_RenderPoints
- SDL_RenderRect
- SDL_RenderRects
- SDL_RenderFillRect
- SDL_RenderFillRects
- SDL_SetRenderDrawBlendMode
- SDL_SetRenderDrawColor

SDL_RenderReadPixels

Read pixels from the current rendering target.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_Surface * SDL_RenderReadPixels(SDL_Renderer *renderer, const
    SDL_Rect *rect);
```

Function Parameters

renderer	the rendering context
rect	an SDL_Rect structure representing the area in pixels relative to the to current viewport, or NULL for the entire viewport

Return Value

Returns a new SDL_Surface on success or NULL on failure; call SDL_GetError() for more information.

Remarks

The returned surface should be freed with SDL_DestroySurface() **WARNING**: This is a very slow operation, and should not be used frequently. If you're using this on the main rendering target, it should be called after rendering and before SDL_RenderPresent().

Version

This function is available since SDL 3.0.0.

SDL_RenderRect

Draw a rectangle on the current rendering target at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderRect(SDL_Renderer *renderer, const SDL_FRect *rect);
```

Function Parameters

renderer	The renderer which should draw a rectangle.
rect	A pointer to the destination rectangle, or NULL to outline the entire rendering target.

Return Value

Returns 0 on success, or -1 on error

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderRects

SDL_RenderRects

Draw some number of rectangles on the current rendering target at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderRects(SDL_Renderer *renderer, const SDL_FRect *rects, int
    count);
```

Function Parameters

renderer	The renderer which should draw multiple rectangles.
rects	A pointer to an array of destination rectangles.
count	The number of rectangles.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderRect

SDL_RenderTexture

Copy a portion of the texture to the current rendering target at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderTexture(SDL_Renderer *renderer, SDL_Texture *texture,
    const SDL_FRect *srcrect, const SDL_FRect *dstrect);
```

Function Parameters

renderer	The renderer which should copy parts of a texture.
texture	The source texture.
srcrect	A pointer to the source rectangle, or NULL for the entire texture.
dstrect	A pointer to the destination rectangle, or NULL for the entire rendering target.

Return Value

Returns 0 on success, or -1 on error

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderTextureRotated

SDL_RenderTextureRotated

Copy a portion of the source texture to the current rendering target, with rotation and flipping, at subpixel precision.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_RenderTextureRotated(SDL_Renderer *renderer, SDL_Texture
    *texture,
                             const SDL_FRect *srcrect, const SDL_FRect
                                 *dstrect,
                             const double angle, const SDL_FPoint *center,
                             const SDL_FlipMode flip);
```

Function Parameters

renderer	The renderer which should copy parts of a texture.
texture	The source texture.
srcrect	A pointer to the source rectangle, or NULL for the entire texture.
dstrect	A pointer to the destination rectangle, or NULL for the entire rendering target.
angle	An angle in degrees that indicates the rotation that will be applied to dstrect, rotating it in a clockwise direction
center	A pointer to a point indicating the point around which dstrect will be rotated (if NULL, rotation will be done around dstrect.w/2, dstrect.h/2).
flip	An SDL_FlipMode value stating which flipping actions should be performed on the texture

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RenderTexture

SDL_RenderViewportSet

Return whether an explicit rectangle was set as the viewport.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
SDL_bool SDL_RenderViewportSet(SDL_Renderer *renderer);
```

Function Parameters

renderer	the rendering context

Return Value

Returns SDL_TRUE if the viewport was set to a specific rectangle, or SDL_FALSE if it was set to NULL (the entire target)

Remarks

This is useful if you're saving and restoring the viewport and want to know whether you should restore a specific rectangle or NULL. Note that the viewport is always reset when changing rendering targets.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderViewport
- SDL_SetRenderViewport

SDL_ReportAssertion

Never call this directly.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
SDL_AssertState SDL_ReportAssertion(SDL_AssertData *data,
                                    const char *func,
                                    const char *file, int line)
#ifdef __clang__
#if __has_feature(attribute_analyzer_noreturn)
   __attribute__((analyzer_noreturn))
#endif
#endif
;
/* Previous 'analyzer_noreturn' attribute tells Clang's static analysis
    that we're a custom assert function,
   and that the analyzer should assume the condition was always true
       past this
   SDL_assert test. */


/* Define the trigger breakpoint call used in asserts */
#ifndef SDL_AssertBreakpoint
#if defined(ANDROID) && defined(assert)
/* Define this as empty in case assert() is defined as SDL_assert */
#define SDL_AssertBreakpoint()
#else
#define SDL_AssertBreakpoint() SDL_TriggerBreakpoint()
#endif
#endif /* !SDL_AssertBreakpoint */

/* the do {} while(0) avoids dangling else problems:
    if (x) SDL_assert(y); else blah();
```

Function Parameters

data	assert data structure
func	function name
file	file name
line	line number

Return Value

Returns assert state

Remarks

Use the SDL_assert* macros instead.

Version

This function is available since SDL 3.0.0.

SDL_ResetAssertionReport

Clear the list of all assertion failures.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
void SDL_ResetAssertionReport(void);
```

Remarks

This function will clear the list of all assertions triggered up to that point. Immediately following this call, SDL_GetAssertionReport will return no items. In addition, any previously-triggered assertions will be reset to a trigger_count of zero, and their always_ignore state will be false.

Code Examples

```
SDL_assert(1+1 == 3); // trigger an assertion.
printf("%p\n", SDL_GetAssertionReport()); // not NULL.
SDL_ResetAssertionReport();
printf("%p\n", SDL_GetAssertionReport()); // NULL.
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAssertionReport

SDL_ResetHint

Reset a hint to the default value.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
SDL_bool SDL_ResetHint(const char *name);
```

Function Parameters

name	the hint to set

Return Value

Returns SDL_TRUE if the hint was set, SDL_FALSE otherwise.

Remarks

This will reset a hint to the value of the environment variable, or NULL if the environment isn't set. Callbacks will be called normally with this change.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetHint
- SDL_ResetHints

SDL_ResetHints

Reset all hints to the default values.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
void SDL_ResetHints(void);
```

Remarks

This will reset all hints to the value of the associated environment variable, or NULL if the environment isn't set. Callbacks will be called normally with this change.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ResetHint

SDL_ResetKeyboard

Clear the state of the keyboard.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
void SDL_ResetKeyboard(void);
```

Remarks

This function will generate key up events for all pressed keys.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetKeyboardState

SDL_RestoreWindow

Request that the size and position of a minimized or maximized window be restored.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_RestoreWindow(SDL_Window *window);
```

Function Parameters

window	the window to restore

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

On some windowing systems this request is asynchronous and the new window state may not have have been applied immediately upon the return of this function. If an immediate change is required, call SDL_SyncWindow() to block until the changes have taken effect.

When the window state changes, an SDL_EVENT_WINDOW_RESTORED event will be emitted. Note that, as this is just a request, the windowing system can deny the state change.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_MaximizeWindow
- SDL_MinimizeWindow
- SDL_SyncWindow

SDL_ResumeAudioDevice

Use this function to unpause audio playback on a specified device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_ResumeAudioDevice(SDL_AudioDeviceID dev);
```

Function Parameters

dev	a device opened by SDL_OpenAudioDevice()

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function unpauses audio processing for a given device that has previously been paused with SDL_PauseAudioDevice(). Once unpaused, any bound audio streams will begin to progress again, and audio can be generated.

Unlike in SDL2, audio devices start in an _unpaused_ state, since an app has to bind a stream before any audio will flow. Unpausing an unpaused device is a legal no-op.

Physical devices can not be paused or unpaused, only logical devices created through SDL_OpenAudioDevice() can be.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AudioDevicePaused
- SDL_PauseAudioDevice

SDL_ResumeHaptic

Resume a haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_ResumeHaptic(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic device to unpause

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Call to unpause after SDL_PauseHaptic().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PauseHaptic

SDL_round

Round `x` to the nearest integer.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_round(double x);
```

Function Parameters

x	floating point value

Return Value

Returns the nearest integer to `x`

Remarks

Rounds `x` to the nearest integer. Values halfway between integers will be rounded away from zero.

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`, y integer

This function operates on double-precision floating point values, use SDL_roundf for single-precision floats. To get the result as an integer type, use SDL_lround.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_roundf
- SDL_lround
- SDL_floor
- SDL_ceil
- SDL_trunc

SDL_roundf

Round `x` to the nearest integer.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_roundf(float x);
```

Function Parameters

x	floating point value

Return Value

Returns the nearest integer to `x`

Remarks

Rounds `x` to the nearest integer. Values halfway between integers will be rounded away from zero.

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`, y integer

This function operates on double-precision floating point values, use SDL_roundf for single-precision floats. To get the result as an integer type, use SDL_lroundf.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_round
- SDL_lroundf
- SDL_floorf
- SDL_ceilf
- SDL_truncf

SDL_RumbleGamepad

Start a rumble effect on a gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_RumbleGamepad(SDL_Gamepad *gamepad, Uint16 low_frequency_rumble,
    Uint16 high_frequency_rumble, Uint32 duration_ms);
```

Function Parameters

gamepad	The gamepad to vibrate
low_frequency_rumble	The intensity of the low frequency (left) rumble motor, from 0 to 0xFFFF
high_frequency_rumble	The intensity of the high frequency (right) rumble motor, from 0 to 0xFFFF
duration_ms	The duration of the rumble effect, in milliseconds

Return Value

Returns 0, or -1 if rumble isn't supported on this gamepad

Remarks

Each call to this function cancels any previous rumble effect, and calling it with 0 intensity stops any rumbling.

This function requires you to process SDL events or call SDL_UpdateJoysticks() to update rumble state.

Version

This function is available since SDL 3.0.0.

SDL_RumbleGamepadTriggers

Start a rumble effect in the gamepad's triggers.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_RumbleGamepadTriggers(SDL_Gamepad *gamepad, Uint16 left_rumble,
    Uint16 right_rumble, Uint32 duration_ms);
```

Function Parameters

gamepad	The gamepad to vibrate
left_rumble	The intensity of the left trigger rumble motor, from 0 to 0xFFFF
right_rumble	The intensity of the right trigger rumble motor, from 0 to 0xFFFF
duration_ms	The duration of the rumble effect, in milliseconds

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Each call to this function cancels any previous trigger rumble effect, and calling it with 0 intensity stops any rumbling.

Note that this is rumbling of the _triggers_ and not the gamepad as a whole. This is currently only supported on Xbox One gamepads. If you want the (more common) whole-gamepad rumble, use SDL_RumbleGamepad() instead.

This function requires you to process SDL events or call SDL_UpdateJoysticks() to update rumble state.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RumbleGamepad

SDL_RumbleJoystick

Start a rumble effect.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_RumbleJoystick(SDL_Joystick *joystick, Uint16
    low_frequency_rumble, Uint16 high_frequency_rumble, Uint32
    duration_ms);
```

Function Parameters

joystick	The joystick to vibrate
low_frequency_rumble	The intensity of the low frequency (left) rumble motor, from 0 to 0xFFFF
high_frequency_rumble	The intensity of the high frequency (right) rumble motor, from 0 to 0xFFFF
duration_ms	The duration of the rumble effect, in milliseconds

Return Value

Returns 0, or -1 if rumble isn't supported on this joystick

Remarks

Each call to this function cancels any previous rumble effect, and calling it with 0 intensity stops any rumbling.

This function requires you to process SDL events or call SDL_UpdateJoysticks() to update rumble state.

Version

This function is available since SDL 3.0.0.

SDL_RumbleJoystickTriggers

Start a rumble effect in the joystick's triggers.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_RumbleJoystickTriggers(SDL_Joystick *joystick, Uint16
    left_rumble, Uint16 right_rumble, Uint32 duration_ms);
```

Function Parameters

joystick	The joystick to vibrate
left_rumble	The intensity of the left trigger rumble motor, from 0 to 0xFFFF
right_rumble	The intensity of the right trigger rumble motor, from 0 to 0xFFFF
duration_ms	The duration of the rumble effect, in milliseconds

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Each call to this function cancels any previous trigger rumble effect, and calling it with 0 intensity stops any rumbling.

Note that this is rumbling of the _triggers_ and not the game controller as a whole. This is currently only supported on Xbox One controllers. If you want the (more common) whole-controller rumble, use SDL_RumbleJoystick() instead.

This function requires you to process SDL events or call SDL_UpdateJoysticks() to update rumble state.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RumbleJoystick

SDL_RunApp

Initializes and launches an SDL application, by doing platform-specific initialization before calling your mainFunction and cleanups after it returns, if that is needed for a specific platform, otherwise it just calls mainFunction.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
int SDL_RunApp(int argc, char* argv[], SDL_main_func mainFunction, void
    * reserved);
```

Function Parameters

argc	The argc parameter from the application's main() function, or 0 if the platform's main-equivalent has no argc
argv	The argv parameter from the application's main() function, or NULL if the platform's main-equivalent has no argv
mainFunction	Your SDL app's C-style main(), an SDL_main_func. NOT the function you're calling this from! Its name doesn't matter, but its signature must be like int my_main(int argc, char* argv[])
reserved	should be NULL (reserved for future use, will probably be platform-specific then)

Return Value

Returns the return value from mainFunction: 0 on success, -1 on failure; SDL_GetError() might have more information on the failure

Remarks

You can use this if you want to use your own main() implementation without using SDL_main (like when using SDL_MAIN_HANDLED). When using this, you do *not* need SDL_SetMainReady().

Version

This function is available since SDL 3.0.0.

SDL_RunHapticEffect

Run the haptic effect on its associated haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_RunHapticEffect(SDL_Haptic *haptic, int effect, Uint32
    iterations);
```

Function Parameters

haptic	the SDL_Haptic device to run the effect on
effect	the ID of the haptic effect to run
iterations	the number of iterations to run the effect; use `SDL_HAPTIC_INFINITY` to repeat forever

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

To repeat the effect over and over indefinitely, set `iterations` to `SDL_HAPTIC_INFINITY`. (Repeats the envelope - attack and fade.) To make one instance of the effect last indefinitely (so the effect does not fade), set the effect's `length` in its structure/union to `SDL_HAPTIC_INFINITY` instead.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetHapticEffectStatus
- SDL_StopHapticEffect
- SDL_StopHapticEffects

SDL_SaveBMP

Save a surface to a file.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SaveBMP(SDL_Surface *surface, const char *file);
```

Function Parameters

surface	the SDL_Surface structure containing the image to be saved
file	a file to save to

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Surfaces with a 24-bit, 32-bit and paletted 8-bit format get saved in the BMP directly. Other RGB formats with 8-bit or higher get converted to a 24-bit surface or, if they have an alpha mask or a colorkey, to a 32-bit surface before they are saved. YUV and paletted 1-bit and 4-bit formats are not supported.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LoadBMP
- SDL_SaveBMP_IO

SDL_SaveBMP_IO

Save a surface to a seekable SDL data stream in BMP format.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SaveBMP_IO(SDL_Surface *surface, SDL_IOStream *dst, SDL_bool
    closeio);
```

Function Parameters

surface	the SDL_Surface structure containing the image to be saved
dst	a data stream to save to
closeio	if SDL_TRUE, calls SDL_CloseIO() on `dst` before returning, even in the case of an error

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Surfaces with a 24-bit, 32-bit and paletted 8-bit format get saved in the BMP directly. Other RGB formats with 8-bit or higher get converted to a 24-bit surface or, if they have an alpha mask or a colorkey, to a 32-bit surface before they are saved. YUV and paletted 1-bit and 4-bit formats are not supported.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LoadBMP_IO
- SDL_SaveBMP

SDL_scalbn

Scale `x` by an integer power of two.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_scalbn(double x, int n);
```

Function Parameters

x	floating point value to be scaled
n	integer exponent

Return Value

Returns `x * 2n̂`

Remarks

Multiplies `x` by the `n`th power of the floating point radix (always 2).

Domain: `-INF <= x <= INF`, `n` integer

Range: `-INF <= y <= INF`

This function operates on double-precision floating point values, use SDL_scalbnf for single-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_scalbnf
- SDL_pow

SDL_scalbnf

Scale `x` by an integer power of two.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_scalbnf(float x, int n);
```

Function Parameters

x	floating point value to be scaled
n	integer exponent

Return Value

Returns `x * 2n̂`

Remarks

Multiplies `x` by the `n`th power of the floating point radix (always 2).

Domain: `-INF <= x <= INF`, `n` integer

Range: `-INF <= y <= INF`

This function operates on single-precision floating point values, use SDL_scalbn for double-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_scalbn
- SDL_powf

SDL_ScreenKeyboardShown

Check whether the screen keyboard is shown for given window.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_bool SDL_ScreenKeyboardShown(SDL_Window *window);
```

Function Parameters

window	the window for which screen keyboard should be queried

Return Value

Returns SDL_TRUE if screen keyboard is shown or SDL_FALSE if not.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HasScreenKeyboardSupport

SDL_ScreenSaverEnabled

Check whether the screensaver is currently enabled.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_bool SDL_ScreenSaverEnabled(void);
```

Return Value

Returns SDL_TRUE if the screensaver is enabled, SDL_FALSE if it is disabled.

Remarks

The screensaver is disabled by default.

The default can also be changed using `SDL_HINT_VIDEO_ALLOW_SCREENSAVER`.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DisableScreenSaver
- SDL_EnableScreenSaver

SDL_SeekIO

Seek within an SDL_IOStream data stream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
Sint64 SDL_SeekIO(SDL_IOStream *context, Sint64 offset, int whence);
```

Function Parameters

context	a pointer to an SDL_IOStream structure
offset	an offset in bytes, relative to **whence** location; can be negative
whence	any of `SDL_IO_SEEK_SET`, `SDL_IO_SEEK_CUR`, `SDL_IO_SEEK_END`

Return Value

Returns the final offset in the data stream after the seek or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function seeks to byte `offset`, relative to `whence`. `whence` may be any of the following values:

- `SDL_IO_SEEK_SET`: seek from the beginning of data
- `SDL_IO_SEEK_CUR`: seek relative to current read point
- `SDL_IO_SEEK_END`: seek relative to the end of data

If this stream can not seek, it will return -1.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_TellIO

SDL_SendGamepadEffect

Send a gamepad specific effect packet.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_SendGamepadEffect(SDL_Gamepad *gamepad, const void *data, int
    size);
```

Function Parameters

gamepad	The gamepad to affect
data	The data to send to the gamepad
size	The size of the data to send to the gamepad

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_SendJoystickEffect

Send a joystick specific effect packet.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_SendJoystickEffect(SDL_Joystick *joystick, const void *data, int
    size);
```

Function Parameters

joystick	The joystick to affect
data	The data to send to the joystick
size	The size of the data to send to the joystick

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_SetAssertionHandler

Set an application-defined assertion handler.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
void SDL_SetAssertionHandler(
                  SDL_AssertionHandler handler,
                  void *userdata);
```

Function Parameters

handler	the SDL_AssertionHandler function to call when an assertion fails or NULL for the default handler
userdata	a pointer that is passed to `handler`

Remarks

This function allows an application to show its own assertion UI and/or force the response to an assertion failure. If the application doesn't provide this, SDL will try to do the right thing, popping up a system-specific GUI dialog, and probably minimizing any fullscreen windows.

This callback may fire from any thread, but it runs wrapped in a mutex, so it will only fire from one thread at a time.

This callback is NOT reset to SDL's internal handler upon SDL_Quit()!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAssertionHandler

SDL_SetAudioPostmixCallback

Set a callback that fires when data is about to be fed to an audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_SetAudioPostmixCallback(SDL_AudioDeviceID devid,
    SDL_AudioPostmixCallback callback, void *userdata);
```

Function Parameters

devid	The ID of an opened audio device.
callback	A callback function to be called. Can be NULL.
userdata	App-controlled pointer passed to callback. Can be NULL.

Return Value

Returns zero on success, -1 on error; call SDL_GetError() for more information.

Remarks

This is useful for accessing the final mix, perhaps for writing a visualizer or applying a final effect to the audio data before playback.

The buffer is the final mix of all bound audio streams on an opened device; this callback will fire regularly for any device that is both opened and unpaused. If there is no new data to mix, either because no streams are bound to the device or all the streams are empty, this callback will still fire with the entire buffer set to silence.

This callback is allowed to make changes to the data; the contents of the buffer after this call is what is ultimately passed along to the hardware.

The callback is always provided the data in float format (values from -1.0f to 1.0f), but the number of channels or sample rate may be different than the format the app requested when opening the device; SDL might have had to manage a conversion behind the scenes, or the playback might have jumped to new physical hardware when a system default changed, etc. These details may change between calls. Accordingly, the size of the buffer might change between calls as well.

This callback can run at any time, and from any thread; if you need to serialize access to your app's data, you should provide and use a mutex or other synchronization device.

All of this to say: there are specific needs this callback can fulfill, but it is not the simplest interface. Apps should generally provide audio in their preferred format through an SDL_AudioStream and let SDL handle the difference.

This function is extremely time-sensitive; the callback should do the least amount of work possible and return as quickly as it can. The longer the callback runs, the higher the risk of audio dropouts or other problems.

This function will block until the audio device is in between iterations, so any existing callback that might be running will finish before this function sets the new callback and returns.

Setting a NULL callback function disables any previously-set callback.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_SetAudioStreamFormat

Change the input and output formats of an audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_SetAudioStreamFormat(SDL_AudioStream *stream,
                             const SDL_AudioSpec *src_spec,
                             const SDL_AudioSpec *dst_spec);
```

Function Parameters

stream	The stream the format is being changed
src_spec	The new format of the audio input; if NULL, it is not changed.
dst_spec	The new format of the audio output; if NULL, it is not changed.

Return Value

Returns 0 on success, or -1 on error.

Remarks

Future calls to and SDL_GetAudioStreamAvailable and SDL_GetAudioStreamData will reflect the new format, and future calls to SDL_PutAudioStreamData must provide data in the new input formats.

Data that was previously queued in the stream will still be operated on in the format that was current when it was added, which is to say you can put the end of a sound file in one format to a stream, change formats for the next sound file, and start putting that new data while the previous sound file is still queued, and everything will still play back correctly.

Thread Safety

It is safe to call this function from any thread, as it holds a stream-specific mutex while running.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAudioStreamFormat
- SDL_SetAudioStreamFrequencyRatio

SDL_SetAudioStreamFrequencyRatio

Change the frequency ratio of an audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_SetAudioStreamFrequencyRatio(SDL_AudioStream *stream, float
    ratio);
```

Function Parameters

stream	The stream the frequency ratio is being changed
ratio	The frequency ratio. 1.0 is normal speed. Must be between 0.01 and 100.

Return Value

Returns 0 on success, or -1 on error.

Remarks

The frequency ratio is used to adjust the rate at which input data is consumed. Changing this effectively modifies the speed and pitch of the audio. A value greater than 1.0 will play the audio faster, and at a higher pitch. A value less than 1.0 will play the audio slower, and at a lower pitch.

This is applied during SDL_GetAudioStreamData, and can be continuously changed to create various effects.

Thread Safety

It is safe to call this function from any thread, as it holds a stream-specific mutex while running.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetAudioStreamFrequencyRatio
- SDL_SetAudioStreamFormat

SDL_SetAudioStreamGetCallback

Set a callback that runs when data is requested from an audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_SetAudioStreamGetCallback(SDL_AudioStream *stream,
    SDL_AudioStreamCallback callback, void *userdata);
```

Function Parameters

stream	the audio stream to set the new callback on.
callback	the new callback function to call when data is added to the stream.
userdata	an opaque pointer provided to the callback for its own personal use.

Return Value

Returns 0 on success, -1 on error. This only fails if `stream` is NULL.

Remarks

This callback is called _before_ data is obtained from the stream, giving the callback the chance to add more on-demand.

The callback can (optionally) call SDL_PutAudioStreamData() to add more audio to the stream during this call; if needed, the request that triggered this callback will obtain the new data immediately.

The callback's `approx_request` argument is roughly how many bytes of _unconverted_ data (in the stream's input format) is needed by the caller, although this may overestimate a little for safety. This takes into account how much is already in the stream and only asks for any extra necessary to resolve the request, which means the callback may be asked for zero bytes, and a different amount on each call.

The callback is not required to supply exact amounts; it is allowed to supply too much or too little or none at all. The caller will get what's available, up to the amount they requested, regardless of this callback's outcome.

Clearing or flushing an audio stream does not call this callback.

This function obtains the stream's lock, which means any existing callback (get or put) in progress will finish running before setting the new callback.

Setting a NULL function turns off the callback.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetAudioStreamPutCallback

SDL_SetAudioStreamPutCallback

Set a callback that runs when data is added to an audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_SetAudioStreamPutCallback(SDL_AudioStream *stream,
    SDL_AudioStreamCallback callback, void *userdata);
```

Function Parameters

stream	the audio stream to set the new callback on.
callback	the new callback function to call when data is added to the stream.
userdata	an opaque pointer provided to the callback for its own personal use.

Return Value

Returns 0 on success, -1 on error. This only fails if `stream` is NULL.

Remarks

This callback is called _after_ the data is added to the stream, giving the callback the chance to obtain it immediately.

The callback can (optionally) call SDL_GetAudioStreamData() to obtain audio from the stream during this call.

The callback's `approx_request` argument is how many bytes of _converted_ data (in the stream's output format) was provided by the caller, although this may underestimate a little for safety. This value might be less than what is currently available in the stream, if data was already there, and might be less than the caller provided if the stream needs to keep a buffer to aid in resampling. Which means the callback may be provided with zero bytes, and a different amount on each call.

The callback may call SDL_GetAudioStreamAvailable to see the total amount currently available to read from the stream, instead of the total provided by the current call.

The callback is not required to obtain all data. It is allowed to read less or none at all. Anything not read now simply remains in the stream for later access.

Clearing or flushing an audio stream does not call this callback.

This function obtains the stream's lock, which means any existing callback (get or put) in progress will finish running before setting the new callback.

Setting a NULL function turns off the callback.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetAudioStreamGetCallback

SDL_SetBooleanProperty

Set a boolean property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_SetBooleanProperty(SDL_PropertiesID props, const char *name,
    SDL_bool value);
```

Function Parameters

props	the properties to modify
name	the name of the property to modify
value	the new value of the property

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetBooleanProperty

SDL_SetClipboardData

Offer clipboard data to the OS.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
int SDL_SetClipboardData(SDL_ClipboardDataCallback callback,
    SDL_ClipboardCleanupCallback cleanup, void *userdata, const char
    **mime_types, size_t num_mime_types);
```

Function Parameters

callback	A function pointer to the function that provides the clipboard data
cleanup	A function pointer to the function that cleans up the clipboard data
userdata	An opaque pointer that will be forwarded to the callbacks
mime_types	A list of mime-types that are being offered
num_mime_types	The number of mime-types in the mime_types list

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Tell the operating system that the application is offering clipboard data for each of the proivded mime-types. Once another application requests the data the callback function will be called allowing it to generate and respond with the data for the requested mime-type.

The size of text data does not include any terminator, and the text does not need to be null terminated (e.g. you can directly copy a portion of a document)

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ClearClipboardData
- SDL_GetClipboardData
- SDL_HasClipboardData

SDL_SetClipboardText

Put UTF-8 text into the clipboard.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
int SDL_SetClipboardText(const char *text);
```

Function Parameters

text	the text to store in the clipboard

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetClipboardText
- SDL_HasClipboardText

SDL_SetCursor

Set the active cursor.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
int SDL_SetCursor(SDL_Cursor * cursor);
```

Function Parameters

cursor	a cursor to make active

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function sets the currently active cursor to the specified one. If the cursor is currently visible, the change will be immediately represented on the display. SDL_SetCursor(NULL) can be used to force cursor redraw, if this is desired for any reason.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetCursor

SDL_SetError

Set the SDL error message for the current thread.

Header File

Defined in SDL3/SDL_error.h

Syntax

```
int SDL_SetError(SDL_PRINTF_FORMAT_STRING const char *fmt, ...)
    SDL_PRINTF_VARARG_FUNC(1);
```

Function Parameters

fmt	a printf()-style message format string
...	additional parameters matching

Return Value

Returns always -1.

Remarks

Calling this function will replace any previous error message that was set.

This function always returns -1, since SDL frequently uses -1 to signify an failing result, leading to this idiom:

```
if (error_code) {
   return SDL_SetError("This operation has failed: %d", error_code);
}
```

Code Examples

```
SDL_SetError("Something unexpected happened!");
```

```
int errorCode = 0;
/* ... */
errorCode = -37;
/* ... */
if (errorCode < 0)
   SDL_SetError("Something unexpected happened: Error Code %d",
       errorCode);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ClearError
- SDL_GetError

SDL_SetEventEnabled

Set the state of processing events by type.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
void SDL_SetEventEnabled(Uint32 type, SDL_bool enabled);
```

Function Parameters

type	the type of event; see SDL_EventType for details
enabled	whether to process the event or not

Version

This function is available since SDL 3.0.0.

See Also

- SDL_EventEnabled

SDL_SetEventFilter

Set up a filter to process all events before they change internal state and are posted to the internal event queue.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
void SDL_SetEventFilter(SDL_EventFilter filter, void *userdata);
```

Function Parameters

filter	An SDL_EventFilter function to call when an event happens
userdata	a pointer that is passed to `filter`

Remarks

If the filter function returns 1 when called, then the event will be added to the internal queue. If it returns 0, then the event will be dropped from the queue, but the internal state will still be updated. This allows selective filtering of dynamically arriving events. **WARNING**: Be very careful of what you do in the event filter function, as it may run in a different thread!

On platforms that support it, if the quit event is generated by an interrupt signal (e.g. pressing Ctrl-C), it will be delivered to the application at the next event poll.

There is one caveat when dealing with the ::SDL_QuitEvent event type. The event filter is only called when the window manager desires to close the application window. If the event filter returns 1, then the window will be closed, otherwise the window will remain open if possible.

Note: Disabled events never make it to the event filter function; see SDL_SetEventEnabled().

Note: If you just want to inspect events without filtering, you should use SDL_AddEventWatch() instead.

Note: Events pushed onto the queue with SDL_PushEvent() get passed through the event filter, but events pushed onto the queue with SDL_PeepEvents() do not.

Code Examples

```
#include <SDL3/SDL.h>
```

```
// Just a quick warning: this is a silly way to do things, but it
//  illustrates how event filters work. In real life, you'd just
//  handle the event when you get it from SDL_PollEvent in the main
//  loop. Generally if you find yourself using an event filter, you
//  should stop and think carefully about whether you have a good
//  reason to be doing so in the first place.


// This is the function we'll pass to SDL_SetEventFilter. If it sees
// that the user has pressed the keyboard's space bar, it toggles
// the value pointed to by `userdata`, to represent the color blue,
// between 255 and 0. The main program uses this value to clear the
// window to a specific color as the space bar is pressed.
static int SDLCALL my_event_filter(void *userdata, SDL_Event * event)
{
    if ((event->type == SDL_EVENT_KEY_DOWN) && (event->key.keysym.sym ==
        SDLK_SPACE)) {
        Uint8 *blue = (Uint8 *) userdata;
        if (*blue == 0) {
            *blue = 255;
        } else {
            *blue = 0;
        }
    }
    return 1; // let all events be added to the queue since we always
        return 1.
}

int main(int argc, char **argv)
{
    Uint8 blue = 0;
    int quit = 0;

    // Just get a window on the screen and clear it to black.
    // In real life, you should check these for errors!
    SDL_Init(SDL_INIT_VIDEO);
    SDL_Window *window = SDL_CreateWindow("Hello SDL", 640, 480, 0);
    SDL_Renderer *renderer = SDL_CreateRenderer(window, NULL,
        SDL_RENDERER_PRESENTVSYNC);
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
    SDL_RenderClear(renderer);
    SDL_RenderPresent(renderer);

    // Set up an event filter...
    SDL_SetEventFilter(my_event_filter, &blue);

    // Now run the event loop mostly forever. Each event goes through the
    //  my_event_filter function before it lands here. Each frame, we
        fill
```

```
    // in the window with whatever color ‘blue‘ is set to, which might
        be
    // changed by the filter function, as its address is our userdata.
    while (!quit) {
        SDL_Event e;
        while (SDL_PollEvent(&e)) {
            if (e.type == SDL_EVENT_QUIT) {
                quit = 1;
            }
        }

        SDL_SetRenderDrawColor(renderer, 0, 0, blue, 255);
        SDL_RenderClear(renderer);
        SDL_RenderPresent(renderer);
    }

    SDL_Quit();
    return 0;
}
```

Thread Safety

SDL may call the filter callback at any time from any thread; the application is responsible for locking resources the callback touches that need to be protected.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AddEventWatch
- SDL_SetEventEnabled
- SDL_GetEventFilter
- SDL_PeepEvents
- SDL_PushEvent

SDL_SetFloatProperty

Set a floating point property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_SetFloatProperty(SDL_PropertiesID props, const char *name, float
    value);
```

Function Parameters

props	the properties to modify
name	the name of the property to modify
value	the new value of the property

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetFloatProperty

SDL_SetGamepadEventsEnabled

Set the state of gamepad event processing.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
void SDL_SetGamepadEventsEnabled(SDL_bool enabled);
```

Function Parameters

enabled	whether to process gamepad events or not

Remarks

If gamepad events are disabled, you must call SDL_UpdateGamepads() yourself and check the state of the gamepad when you want gamepad information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GamepadEventsEnabled
- SDL_UpdateGamepads

SDL_SetGamepadLED

Update a gamepad's LED color.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_SetGamepadLED(SDL_Gamepad *gamepad, Uint8 red, Uint8 green,
    Uint8 blue);
```

Function Parameters

gamepad	The gamepad to update
red	The intensity of the red LED
green	The intensity of the green LED
blue	The intensity of the blue LED

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

An example of a joystick LED is the light on the back of a PlayStation 4's DualShock 4 controller.

For gamepads with a single color LED, the maximum of the RGB values will be used as the LED brightness.

Version

This function is available since SDL 3.0.0.

SDL_SetGamepadMapping

Set the current mapping of a joystick or gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_SetGamepadMapping(SDL_JoystickID instance_id, const char
    *mapping);
```

Function Parameters

instance_id	the joystick instance ID
mapping	the mapping to use for this device, or NULL to clear the mapping

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Details about mappings are discussed with SDL_AddGamepadMapping().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_AddGamepadMapping
- SDL_GetGamepadMapping

SDL_SetGamepadPlayerIndex

Set the player index of an opened gamepad.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_SetGamepadPlayerIndex(SDL_Gamepad *gamepad, int player_index);
```

Function Parameters

gamepad	the gamepad object to adjust.
player_index	Player index to assign to this gamepad, or -1 to clear the player index and turn off player LEDs.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetGamepadPlayerIndex

SDL_SetGamepadSensorEnabled

Set whether data reporting for a gamepad sensor is enabled.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
int SDL_SetGamepadSensorEnabled(SDL_Gamepad *gamepad, SDL_SensorType
    type, SDL_bool enabled);
```

Function Parameters

gamepad	The gamepad to update
type	The type of sensor to enable/disable
enabled	Whether data reporting should be enabled

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GamepadHasSensor
- SDL_GamepadSensorEnabled

SDL_SetHapticAutocenter

Set the global autocenter of the device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_SetHapticAutocenter(SDL_Haptic *haptic, int autocenter);
```

Function Parameters

haptic	the SDL_Haptic device to set autocentering on
autocenter	value to set autocenter to (0-100)

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Autocenter should be between 0 and 100. Setting it to 0 will disable autocentering.

Device must support the SDL_HAPTIC_AUTOCENTER feature.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetHapticFeatures

SDL_SetHapticGain

Set the global gain of the specified haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_SetHapticGain(SDL_Haptic *haptic, int gain);
```

Function Parameters

haptic	the SDL_Haptic device to set the gain on
gain	value to set the gain to, should be between 0 and 100 (0 - 100)

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Device must support the SDL_HAPTIC_GAIN feature.

The user may specify the maximum gain by setting the environment variable `SDL_HAPTIC_GAIN_MAX` which should be between 0 and 100. All calls to SDL_SetHapticGain() will scale linearly using `SDL_HAPTIC_GAIN_MAX` as the maximum.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetHapticFeatures

SDL_SetHint

Set a hint with normal priority.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
SDL_bool SDL_SetHint(const char *name,
                     const char *value);
```

Function Parameters

name	the hint to set
value	the value of the hint variable

Return Value

Returns SDL_TRUE if the hint was set, SDL_FALSE otherwise.

Remarks

Hints will not be set if there is an existing override hint or environment variable that takes precedence. You can use SDL_SetHintWithPriority() to set the hint with override priority instead.

Code Examples

```
/* Log most events (other than the really spammy ones) */
SDL_SetHint(SDL_HINT_LOGGING, "2");
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetHint
- SDL_ResetHint
- SDL_SetHintWithPriority

SDL_SetHintWithPriority

Set a hint with a specific priority.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
SDL_bool SDL_SetHintWithPriority(const char *name,
                                 const char *value,
                                 SDL_HintPriority priority);
```

Function Parameters

name	the hint to set
value	the value of the hint variable
priority	the SDL_HintPriority level for the hint

Return Value

Returns SDL_TRUE if the hint was set, SDL_FALSE otherwise.

Remarks

The priority controls the behavior when setting a hint that already has a value. Hints will replace existing hints of their priority and lower. Environment variables are considered to have override priority.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetHint
- SDL_ResetHint
- SDL_SetHint

SDL_SetJoystickEventsEnabled

Set the state of joystick event processing.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
void SDL_SetJoystickEventsEnabled(SDL_bool enabled);
```

Function Parameters

enabled	whether to process joystick events or not

Remarks

If joystick events are disabled, you must call SDL_UpdateJoysticks() yourself and check the state of the joystick when you want joystick information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_JoystickEventsEnabled
- SDL_UpdateJoysticks

SDL_SetJoystickLED

Update a joystick's LED color.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_SetJoystickLED(SDL_Joystick *joystick, Uint8 red, Uint8 green,
    Uint8 blue);
```

Function Parameters

joystick	The joystick to update
red	The intensity of the red LED
green	The intensity of the green LED
blue	The intensity of the blue LED

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

An example of a joystick LED is the light on the back of a PlayStation 4's DualShock 4 controller.

For joysticks with a single color LED, the maximum of the RGB values will be used as the LED brightness.

Version

This function is available since SDL 3.0.0.

SDL_SetJoystickPlayerIndex

Set the player index of an opened joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_SetJoystickPlayerIndex(SDL_Joystick *joystick, int player_index);
```

Function Parameters

joystick	the SDL_Joystick obtained from SDL_OpenJoystick()
player_index	Player index to assign to this joystick, or -1 to clear the player index and turn off player LEDs.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetJoystickPlayerIndex

SDL_SetJoystickVirtualAxis

Set values on an opened, virtual-joystick's axis.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_SetJoystickVirtualAxis(SDL_Joystick *joystick, int axis, Sint16
    value);
```

Function Parameters

joystick	the virtual joystick on which to set state.
axis	the specific axis on the virtual joystick to set.
value	the new value for the specified axis.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Please note that values set here will not be applied until the next call to SDL_UpdateJoysticks, which can either be called directly, or can be called indirectly through various other SDL APIs, including, but not limited to the following: SDL_PollEvent, SDL_PumpEvents, SDL_WaitEventTimeout, SDL_WaitEvent.

Note that when sending trigger axes, you should scale the value to the full range of Sint16. For example, a trigger at rest would have the value of `SDL_JOYSTICK_AXIS_MIN`.

Version

This function is available since SDL 3.0.0.

SDL_SetJoystickVirtualButton

Set values on an opened, virtual-joystick's button.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_SetJoystickVirtualButton(SDL_Joystick *joystick, int button,
    Uint8 value);
```

Function Parameters

joystick	the virtual joystick on which to set state.
button	the specific button on the virtual joystick to set.
value	the new value for the specified button.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Please note that values set here will not be applied until the next call to SDL_UpdateJoysticks, which can either be called directly, or can be called indirectly through various other SDL APIs, including, but not limited to the following: SDL_PollEvent, SDL_PumpEvents, SDL_WaitEventTimeout, SDL_WaitEvent.

Version

This function is available since SDL 3.0.0.

SDL_SetJoystickVirtualHat

Set values on an opened, virtual-joystick's hat.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
int SDL_SetJoystickVirtualHat(SDL_Joystick *joystick, int hat, Uint8
    value);
```

Function Parameters

joystick	the virtual joystick on which to set state.
hat	the specific hat on the virtual joystick to set.
value	the new value for the specified hat.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Please note that values set here will not be applied until the next call to SDL_UpdateJoysticks, which can either be called directly, or can be called indirectly through various other SDL APIs, including, but not limited to the following: SDL_PollEvent, SDL_PumpEvents, SDL_WaitEventTimeout, SDL_WaitEvent.

Version

This function is available since SDL 3.0.0.

SDL_SetLogOutputFunction

Replace the default log output function with one of your own.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
void SDL_SetLogOutputFunction(SDL_LogOutputFunction callback, void
    *userdata);
```

Function Parameters

callback	an SDL_LogOutputFunction to call instead of the default
userdata	a pointer that is passed to `callback`

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetLogOutputFunction

SDL_SetMainReady

Circumvent failure of SDL_Init() when not using SDL_main() as an entry point.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
void SDL_SetMainReady(void);
```

Remarks

This function is defined in SDL_main.h, along with the preprocessor rule to redefine main() as SDL_main(). Thus to ensure that your main() function will not be changed it is necessary to define SDL_MAIN_HANDLED before including SDL.h.

Code Examples

```
#define SDL_MAIN_HANDLED
#include <SDL3/SDL.h>
#include <SDL3/SDL_main.h>

int main(int argc, char *argv[])
{
    SDL_SetMainReady();
    SDL_Init(SDL_INIT_VIDEO);

    /* ... */

    SDL_Quit();

    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Init

SDL_SetMemoryFunctions

Replace SDL's memory allocation functions with a custom set.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_SetMemoryFunctions(SDL_malloc_func malloc_func,
                           SDL_calloc_func calloc_func,
                           SDL_realloc_func realloc_func,
                           SDL_free_func free_func);
```

Function Parameters

malloc_func	custom malloc function
calloc_func	custom calloc function
realloc_func	custom realloc function
free_func	custom free function

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_SetModState

Set the current key modifier state for the keyboard.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
void SDL_SetModState(SDL_Keymod modstate);
```

Function Parameters

modstate	the desired SDL_Keymod for the keyboard

Remarks

The inverse of SDL_GetModState(), SDL_SetModState() allows you to impose modifier key states on your application. Simply pass your desired modifier states into `modstate`. This value may be a bitwise, OR'd combination of SDL_Keymod values.

This does not change the keyboard state, only the key modifier flags that SDL reports.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetModState

SDL_SetNumberProperty

Set an integer property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_SetNumberProperty(SDL_PropertiesID props, const char *name,
    Sint64 value);
```

Function Parameters

props	the properties to modify
name	the name of the property to modify
value	the new value of the property

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetNumberProperty

SDL_SetPaletteColors

Set a range of colors in a palette.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
int SDL_SetPaletteColors(SDL_Palette * palette,
                         const SDL_Color * colors,
                         int firstcolor, int ncolors);
```

Function Parameters

palette	the SDL_Palette structure to modify
colors	an array of SDL_Color structures to copy into the palette
firstcolor	the index of the first palette entry to modify
ncolors	the number of entries to modify

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_SetPixelFormatPalette

Set the palette for a pixel format structure.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
int SDL_SetPixelFormatPalette(SDL_PixelFormat * format,
                              SDL_Palette *palette);
```

Function Parameters

format	the SDL_PixelFormat structure that will use the palette
palette	the SDL_Palette structure that will be used

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_SetPrimarySelectionText

Put UTF-8 text into the primary selection.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
int SDL_SetPrimarySelectionText(const char *text);
```

Function Parameters

text	the text to store in the primary selection

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPrimarySelectionText
- SDL_HasPrimarySelectionText

SDL_SetProperty

Set a property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_SetProperty(SDL_PropertiesID props, const char *name, void
    *value);
```

Function Parameters

props	the properties to modify
name	the name of the property to modify
value	the new value of the property, or NULL to delete the property

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_HasProperty
- SDL_SetBooleanProperty
- SDL_SetFloatProperty
- SDL_SetNumberProperty
- SDL_SetPropertyWithCleanup
- SDL_SetStringProperty

SDL_SetPropertyWithCleanup

Set a property on a set of properties with a cleanup function that is called when the property is deleted.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_SetPropertyWithCleanup(SDL_PropertiesID props, const char *name,
    void *value, void (SDLCALL *cleanup)(void *userdata, void *value),
    void *userdata);
```

Function Parameters

props	the properties to modify
name	the name of the property to modify
value	the new value of the property, or NULL to delete the property
cleanup	the function to call when this property is deleted, or NULL if no cleanup is necessary
userdata	a pointer that is passed to the cleanup function

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The cleanup function is also called if setting the property fails for any reason.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetProperty
- SDL_SetProperty

SDL_SetRelativeMouseMode

Set relative mouse mode.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
int SDL_SetRelativeMouseMode(SDL_bool enabled);
```

Function Parameters

enabled	SDL_TRUE to enable relative mode, SDL_FALSE to disable.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

While the mouse is in relative mode, the cursor is hidden, the mouse position is constrained to the window, and SDL will report continuous relative mouse motion even if the mouse is at the edge of the window.

This function will flush any pending mouse motion.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRelativeMouseMode

SDL_SetRenderClipRect

Set the clip rectangle for rendering on the specified target.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderClipRect(SDL_Renderer *renderer, const SDL_Rect *rect);
```

Function Parameters

renderer	the rendering context
rect	an SDL_Rect structure representing the clip area, relative to the viewport, or NULL to disable clipping

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderClipRect
- SDL_RenderClipEnabled

SDL_SetRenderColorScale

Set the color scale used for render operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderColorScale(SDL_Renderer *renderer, float scale);
```

Function Parameters

renderer	the rendering context
scale	the color scale value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The color scale is an additional scale multiplied into the pixel color value while rendering. This can be used to adjust the brightness of colors during HDR rendering, or changing HDR video brightness when playing on an SDR display.

The color scale does not affect the alpha channel, only the color brightness.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderColorScale

SDL_SetRenderDrawBlendMode

Set the blend mode used for drawing operations (Fill and Line).

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderDrawBlendMode(SDL_Renderer *renderer, SDL_BlendMode
    blendMode);
```

Function Parameters

renderer	the rendering context
blendMode	the SDL_BlendMode to use for blending

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If the blend mode is not supported, the closest supported mode is chosen.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderDrawBlendMode

SDL_SetRenderDrawColor

Set the color used for drawing operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderDrawColor(SDL_Renderer *renderer, Uint8 r, Uint8 g,
    Uint8 b, Uint8 a);
```

Function Parameters

renderer	the rendering context
r	the red value used to draw on the rendering target
g	the green value used to draw on the rendering target
b	the blue value used to draw on the rendering target
a	the alpha value used to draw on the rendering target; usually `SDL_ALPHA_OPAQUE` (255). Use SDL_SetRenderDrawBlendMode to specify how the alpha channel is used

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Set the color for drawing or filling rectangles, lines, and points, and for SDL_RenderClear().

Code Examples

```
SDL_Renderer *renderer;
SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);

SDL_FRect rectangle;
rectangle.x = 0.f;
rectangle.y = 0.f;
rectangle.w = 50.f;
rectangle.h = 50.f;

SDL_RenderFillRect(renderer, &rectangle);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderDrawColor
- SDL_SetRenderDrawColorFloat

SDL_SetRenderDrawColorFloat

Set the color used for drawing operations (Rect, Line and Clear).

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderDrawColorFloat(SDL_Renderer *renderer, float r, float
    g, float b, float a);
```

Function Parameters

renderer	the rendering context
r	the red value used to draw on the rendering target
g	the green value used to draw on the rendering target
b	the blue value used to draw on the rendering target
a	the alpha value used to draw on the rendering target. Use SDL_SetRenderDrawBlendMode to specify how the alpha channel is used

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Set the color for drawing or filling rectangles, lines, and points, and for SDL_RenderClear().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderDrawColorFloat
- SDL_SetRenderDrawColor

SDL_SetRenderLogicalPresentation

Set a device independent resolution and presentation mode for rendering.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderLogicalPresentation(SDL_Renderer *renderer, int w, int
    h, SDL_RendererLogicalPresentation mode, SDL_ScaleMode scale_mode);
```

Function Parameters

renderer	the rendering context
w	the width of the logical resolution
h	the height of the logical resolution
mode	the presentation mode used
scale_mode	the scale mode used

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function sets the width and height of the logical rendering output. A render target is created at the specified size and used for rendering and then copied to the output during presentation.

You can disable logical coordinates by setting the mode to SDL_LOGICAL_PRESENTATION_DISABLED, and in that case you get the full pixel resolution of the output window.

You can convert coordinates in an event into rendering coordinates using SDL_ConvertEventToRenderCoordinates().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ConvertEventToRenderCoordinates
- SDL_GetRenderLogicalPresentation

SDL_SetRenderScale

Set the drawing scale for rendering on the current target.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderScale(SDL_Renderer *renderer, float scaleX, float
    scaleY);
```

Function Parameters

renderer	the rendering context
scaleX	the horizontal scaling factor
scaleY	the vertical scaling factor

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The drawing coordinates are scaled by the x/y scaling factors before they are used by the renderer. This allows resolution independent drawing with a single coordinate system.

If this results in scaling or subpixel drawing by the rendering backend, it will be handled using the appropriate quality hints. For best results use integer scaling factors.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderScale

SDL_SetRenderTarget

Set a texture as the current rendering target.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderTarget(SDL_Renderer *renderer, SDL_Texture *texture);
```

Function Parameters

renderer	the rendering context
texture	the targeted texture, which must be created with the `SDL_TEXTUREACCESS_TARGET` flag, or NULL to render to the window instead of a texture.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The default render target is the window for which the renderer was created. To stop rendering to a texture and render to the window again, call this function with a NULL `texture`.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderTarget

SDL_SetRenderViewport

Set the drawing area for rendering on the current target.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderViewport(SDL_Renderer *renderer, const SDL_Rect *rect);
```

Function Parameters

renderer	the rendering context
rect	the SDL_Rect structure representing the drawing area, or NULL to set the viewport to the entire target

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderViewport
- SDL_RenderViewportSet

SDL_SetRenderVSync

Toggle VSync of the given renderer.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetRenderVSync(SDL_Renderer *renderer, int vsync);
```

Function Parameters

renderer	The renderer to toggle
vsync	1 for on, 0 for off. All other values are reserved

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetRenderVSync

SDL_SetStringProperty

Set a string property on a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
int SDL_SetStringProperty(SDL_PropertiesID props, const char *name,
    const char *value);
```

Function Parameters

props	the properties to modify
name	the name of the property to modify
value	the new value of the property, or NULL to delete the property

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function makes a copy of the string; the caller does not have to preserve the data after this call completes.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetStringProperty

SDL_SetSurfaceAlphaMod

Set an additional alpha value used in blit operations.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SetSurfaceAlphaMod(SDL_Surface *surface, Uint8 alpha);
```

Function Parameters

surface	the SDL_Surface structure to update
alpha	the alpha value multiplied into blit operations

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

When this surface is blitted, during the blit operation the source alpha value is modulated by this alpha value according to the following formula: `srcA = srcA * (alpha / 255)`

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetSurfaceAlphaMod
- SDL_SetSurfaceColorMod

SDL_SetSurfaceBlendMode

Set the blend mode used for blit operations.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SetSurfaceBlendMode(SDL_Surface *surface, SDL_BlendMode
    blendMode);
```

Function Parameters

surface	the SDL_Surface structure to update
blendMode	the SDL_BlendMode to use for blit blending

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

To copy a surface to another surface (or texture) without blending with the existing data, the blendmode of the SOURCE surface should be set to `SDL_BLENDMODE_NONE`.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetSurfaceBlendMode

SDL_SetSurfaceClipRect

Set the clipping rectangle for a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_bool SDL_SetSurfaceClipRect(SDL_Surface *surface, const SDL_Rect
    *rect);
```

Function Parameters

surface	the SDL_Surface structure to be clipped
rect	the SDL_Rect structure representing the clipping rectangle, or NULL to disable clipping

Return Value

Returns SDL_TRUE if the rectangle intersects the surface, otherwise SDL_FALSE and blits will be completely clipped.

Remarks

When `surface` is the destination of a blit, only the area within the clip rectangle is drawn into.

Note that blits are automatically clipped to the edges of the source and destination surfaces.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetSurfaceClipRect

SDL_SetSurfaceColorKey

Set the color key (transparent pixel) in a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SetSurfaceColorKey(SDL_Surface *surface, int flag, Uint32 key);
```

Function Parameters

surface	the SDL_Surface structure to update
flag	SDL_TRUE to enable color key, SDL_FALSE to disable color key
key	the transparent pixel

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The color key defines a pixel value that will be treated as transparent in a blit. For example, one can use this to specify that cyan pixels should be considered transparent, and therefore not rendered.

It is a pixel of the format used by the surface, as generated by SDL_MapRGB().

RLE acceleration can substantially speed up blitting of images with large horizontal runs of transparent pixels. See SDL_SetSurfaceRLE() for details.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetSurfaceColorKey
- SDL_SurfaceHasColorKey

SDL_SetSurfaceColorMod

Set an additional color value multiplied into blit operations.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SetSurfaceColorMod(SDL_Surface *surface, Uint8 r, Uint8 g, Uint8
    b);
```

Function Parameters

surface	the SDL_Surface structure to update
r	the red color value multiplied into blit operations
g	the green color value multiplied into blit operations
b	the blue color value multiplied into blit operations

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

When this surface is blitted, during the blit operation each source color channel is modulated by the appropriate color value according to the following formula: `srcC = srcC * (color / 255)`

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetSurfaceColorMod
- SDL_SetSurfaceAlphaMod

SDL_SetSurfaceColorspace

Set the colorspace used by a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SetSurfaceColorspace(SDL_Surface *surface, SDL_Colorspace
    colorspace);
```

Function Parameters

surface	the SDL_Surface structure to update
colorspace	an SDL_ColorSpace value describing the surface colorspace

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Setting the colorspace doesn't change the pixels, only how they are interpreted in color operations.

Version

This function is available since SDL 3.0.0.

SDL_SetSurfacePalette

Set the palette used by a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SetSurfacePalette(SDL_Surface *surface, SDL_Palette *palette);
```

Function Parameters

surface	the SDL_Surface structure to update
palette	the SDL_Palette structure to use

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

A single palette can be shared with many surfaces.

Version

This function is available since SDL 3.0.0.

SDL_SetSurfaceRLE

Set the RLE acceleration hint for a surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SetSurfaceRLE(SDL_Surface *surface, int flag);
```

Function Parameters

surface	the SDL_Surface structure to optimize
flag	0 to disable, non-zero to enable RLE acceleration

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If RLE is enabled, color key and alpha blending blits are much faster, but the surface must be locked before directly accessing the pixels.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BlitSurface
- SDL_LockSurface
- SDL_UnlockSurface

SDL_SetTextInputRect

Set the rectangle used to type Unicode text inputs.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
int SDL_SetTextInputRect(const SDL_Rect *rect);
```

Function Parameters

rect	the SDL_Rect structure representing the rectangle to receive text (ignored if NULL)

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Native input methods will place a window with word suggestions near it, without covering the text being inputted.

To start text input in a given location, this function is intended to be called before SDL_StartTextInput, although some platforms support moving the rectangle even while text input (and a composition) is active.

Note: If you want to use the system native IME window, try setting hint **SDL_HINT_IME_SHOW_UI** to **1**, otherwise this function won't give you any feedback.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StartTextInput

SDL_SetTextureAlphaMod

Set an additional alpha value multiplied into render copy operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetTextureAlphaMod(SDL_Texture *texture, Uint8 alpha);
```

Function Parameters

texture	the texture to update
alpha	the source alpha value multiplied into copy operations

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

When this texture is rendered, during the copy operation the source alpha value is modulated by this alpha value according to the following formula: `srcA = srcA * (alpha / 255)`

Alpha modulation is not always supported by the renderer; it will return -1 if alpha modulation is not supported.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureAlphaMod
- SDL_SetTextureAlphaModFloat
- SDL_SetTextureColorMod

SDL_SetTextureAlphaModFloat

Set an additional alpha value multiplied into render copy operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetTextureAlphaModFloat(SDL_Texture *texture, float alpha);
```

Function Parameters

texture	the texture to update
alpha	the source alpha value multiplied into copy operations

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

When this texture is rendered, during the copy operation the source alpha value is modulated by this alpha value according to the following formula: `srcA = srcA * alpha`

Alpha modulation is not always supported by the renderer; it will return -1 if alpha modulation is not supported.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureAlphaModFloat
- SDL_SetTextureAlphaMod
- SDL_SetTextureColorModFloat

SDL_SetTextureBlendMode

Set the blend mode for a texture, used by SDL_RenderTexture().

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetTextureBlendMode(SDL_Texture *texture, SDL_BlendMode
    blendMode);
```

Function Parameters

texture	the texture to update
blendMode	the SDL_BlendMode to use for texture blending

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If the blend mode is not supported, the closest supported mode is chosen and this function returns -1.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureBlendMode

SDL_SetTextureColorMod

Set an additional color value multiplied into render copy operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetTextureColorMod(SDL_Texture *texture, Uint8 r, Uint8 g, Uint8
    b);
```

Function Parameters

texture	the texture to update
r	the red color value multiplied into copy operations
g	the green color value multiplied into copy operations
b	the blue color value multiplied into copy operations

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

When this texture is rendered, during the copy operation each source color channel is modulated by the appropriate color value according to the following formula: `srcC = srcC * (color / 255)`

Color modulation is not always supported by the renderer; it will return -1 if color modulation is not supported.

Code Examples

```
SDL_Renderer *renderer;
SDL_Surface *loadedSurface;
/* ... */
SDL_Texture* pTexture = SDL_CreateTextureFromSurface( renderer,
    loadedSurface );
SDL_SetTextureColorMod( pTexture, 64, 64, 64 );
/* ... */
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureColorMod
- SDL_SetTextureAlphaMod
- SDL_SetTextureColorModFloat

SDL_SetTextureColorModFloat

Set an additional color value multiplied into render copy operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetTextureColorModFloat(SDL_Texture *texture, float r, float g,
    float b);
```

Function Parameters

texture	the texture to update
r	the red color value multiplied into copy operations
g	the green color value multiplied into copy operations
b	the blue color value multiplied into copy operations

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

When this texture is rendered, during the copy operation each source color channel is modulated by the appropriate color value according to the following formula: `srcC = srcC * color`

Color modulation is not always supported by the renderer; it will return -1 if color modulation is not supported.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureColorModFloat
- SDL_SetTextureAlphaModFloat
- SDL_SetTextureColorMod

SDL_SetTextureScaleMode

Set the scale mode used for texture scale operations.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_SetTextureScaleMode(SDL_Texture *texture, SDL_ScaleMode
    scaleMode);
```

Function Parameters

texture	The texture to update.
scaleMode	the SDL_ScaleMode to use for texture scaling.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The default texture scale mode is SDL_SCALEMODE_LINEAR.

If the scale mode is not supported, the closest supported mode is chosen.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTextureScaleMode

SDL_SetThreadPriority

Set the priority for the current thread.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
int SDL_SetThreadPriority(SDL_ThreadPriority priority);
```

Function Parameters

priority	the SDL_ThreadPriority to set

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Note that some platforms will not let you alter the priority (or at least, promote the thread to a higher priority) at all, and some require you to be an administrator account. Be prepared for this to fail.

Version

This function is available since SDL 3.0.0.

SDL_SetTLS

Set the current thread's value associated with a thread local storage ID.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
int SDL_SetTLS(SDL_TLSID id, const void *value, void (SDLCALL
    *destructor)(void*));
```

Function Parameters

id	the thread local storage ID
value	the value to associate with the ID for the current thread
destructor	a function called when the thread exits, to free the value

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The function prototype for `destructor` is:

```
void destructor(void *value)
```

where its parameter `value` is what was passed as `value` to SDL_SetTLS().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetTLS

SDL_SetWindowAlwaysOnTop

Set the window to always be above the others.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowAlwaysOnTop(SDL_Window *window, SDL_bool on_top);
```

Function Parameters

window	The window of which to change the always on top state
on_top	SDL_TRUE to set the window always on top, SDL_FALSE to disable

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This will add or remove the window's `SDL_WINDOW_ALWAYS_ON_TOP` flag. This will bring the window to the front and keep the window above the rest.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowFlags

SDL_SetWindowBordered

Set the border state of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowBordered(SDL_Window *window, SDL_bool bordered);
```

Function Parameters

window	the window of which to change the border state
bordered	SDL_FALSE to remove border, SDL_TRUE to add border

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This will add or remove the window's `SDL_WINDOW_BORDERLESS` flag and add or remove the border from the actual window. This is a no-op if the window's border already matches the requested state.

You can't change the border state of a fullscreen window.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowFlags

SDL_SetWindowFocusable

Set whether the window may have input focus.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowFocusable(SDL_Window *window, SDL_bool focusable);
```

Function Parameters

window	the window to set focusable state
focusable	SDL_TRUE to allow input focus, SDL_FALSE to not allow input focus

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_SetWindowFullscreen

Request that the window's fullscreen state be changed.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowFullscreen(SDL_Window *window, SDL_bool fullscreen);
```

Function Parameters

window	the window to change
fullscreen	SDL_TRUE for fullscreen mode, SDL_FALSE for windowed mode

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

By default a window in fullscreen state uses borderless fullscreen desktop mode, but a specific exclusive display mode can be set using SDL_SetWindowFullscreenMode().

On some windowing systems this request is asynchronous and the new fullscreen state may not have have been applied immediately upon the return of this function. If an immediate change is required, call SDL_SyncWindow() to block until the changes have taken effect.

When the window state changes, an SDL_EVENT_WINDOW_ENTER_FULLSCREEN or SDL_EVENT_WINDOW_LEAVE_FULLSCREEN event will be emitted. Note that, as this is just a request, it can be denied by the windowing system.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowFullscreenMode
- SDL_SetWindowFullscreenMode
- SDL_SyncWindow

SDL_SetWindowFullscreenMode

Set the display mode to use when a window is visible and fullscreen.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowFullscreenMode(SDL_Window *window, const
    SDL_DisplayMode *mode);
```

Function Parameters

window	the window to affect
mode	a pointer to the display mode to use, which can be NULL for borderless fullscreen desktop mode, or one of the fullscreen modes returned by SDL_GetFullscreenDisplayModes() to set an exclusive fullscreen mode.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This only affects the display mode used when the window is fullscreen. To change the window size when the window is not fullscreen, use SDL_SetWindowSize().

If the window is currently in the fullscreen state, this request is asynchronous on some windowing systems and the new mode dimensions may not be applied immediately upon the return of this function. If an immediate change is required, call SDL_SyncWindow() to block until the changes have taken effect.

When the new mode takes effect, an SDL_EVENT_WINDOW_RESIZED and/or an SDL_EVENT_WINDOOW_PIXEL_SIZE_CHANGED event will be emitted with the new mode dimensions.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowFullscreenMode
- SDL_SetWindowFullscreen
- SDL_SyncWindow

SDL_SetWindowHitTest

Provide a callback that decides if a window region has special properties.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowHitTest(SDL_Window *window, SDL_HitTest callback, void
    *callback_data);
```

Function Parameters

window	the window to set hit-testing on
callback	the function to call when doing a hit-test
callback_data	an app-defined void pointer passed to **callback**

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Normally windows are dragged and resized by decorations provided by the system window manager (a title bar, borders, etc), but for some apps, it makes sense to drag them from somewhere else inside the window itself; for example, one might have a borderless window that wants to be draggable from any part, or simulate its own title bar, etc.

This function lets the app provide a callback that designates pieces of a given window as special. This callback is run during event processing if we need to tell the OS to treat a region of the window specially; the use of this callback is known as "hit testing."

Mouse input may not be delivered to your application if it is within a special area; the OS will often apply that input to moving the window or resizing the window and not deliver it to the application.

Specifying NULL for a callback disables hit-testing. Hit-testing is disabled by default.

Platforms that don't support this functionality will return -1 unconditionally, even if you're attempting to disable hit-testing.

Your callback may fire at any time, and its firing does not indicate any specific behavior (for example, on Windows, this certainly might fire when the

OS is deciding whether to drag your window, but it fires for lots of other reasons, too, some unrelated to anything you probably care about _and when the mouse isn't actually at the location it is testing_). Since this can fire at any time, you should try to keep your callback efficient, devoid of allocations, etc.

Version

This function is available since SDL 3.0.0.

SDL_SetWindowIcon

Set the icon for a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowIcon(SDL_Window *window, SDL_Surface *icon);
```

Function Parameters

window	the window to change
icon	an SDL_Surface structure containing the icon for the window

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Code Examples

```
#include <SDL3/SDL.h>

int main(int argc, char* argv[]){

  SDL_Event e;

  SDL_Init(SDL_INIT_VIDEO); // Initialize SDL3

  SDL_Window *window = SDL_CreateWindow( // Open a new window
    "SDL2 window icon demo", 640, 480, SDL_WINDOW_RESIZABLE
  );

  SDL_Surface *surface;  // Declare an SDL_Surface to be filled in with
      pixel data from an image file
  Uint16 pixels[16*16] = { // ...or with raw pixel data:
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff,
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff,
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff,
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff,
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff,
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff,
```

```
    0x0fff, 0x0aab, 0x0789, 0x0bcc, 0x0eee, 0x09aa, 0x099a, 0x0ddd,
    0x0fff, 0x0eee, 0x0899, 0x0fff, 0x0fff, 0x1fff, 0x0dde, 0x0dee,
    0x0fff, 0xabbc, 0xf779, 0x8cdd, 0x3fff, 0x9bbc, 0xaaab, 0x6fff,
    0x0fff, 0x3fff, 0xbaab, 0x0fff, 0x0fff, 0x6689, 0x6fff, 0x0dee,
    0xe678, 0xf134, 0x8abb, 0xf235, 0xf678, 0xf013, 0xf568, 0xf001,
    0xd889, 0x7abc, 0xf001, 0x0fff, 0x0fff, 0x0bcc, 0x9124, 0x5fff,
    0xf124, 0xf356, 0x3eee, 0x0fff, 0x7bbc, 0xf124, 0x0789, 0x2fff,
    0xf002, 0xd789, 0xf024, 0x0fff, 0x0fff, 0x0002, 0x0134, 0xd79a,
    0x1fff, 0xf023, 0xf000, 0xf124, 0xc99a, 0xf024, 0x0567, 0x0fff,
    0xf002, 0xe678, 0xf013, 0x0fff, 0x0ddd, 0x0fff, 0x0fff, 0xb689,
    0x8abb, 0x0fff, 0x0fff, 0xf001, 0xf235, 0xf013, 0x0fff, 0xd789,
    0xf002, 0x9899, 0xf001, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0xe789,
    0xf023, 0xf000, 0xf001, 0xe456, 0x8bcc, 0xf013, 0xf002, 0xf012,
    0x1767, 0x5aaa, 0xf013, 0xf001, 0xf000, 0x0fff, 0x7fff, 0xf124,
    0x0fff, 0x089a, 0x0578, 0x0fff, 0x089a, 0x0013, 0x0245, 0x0eff,
    0x0223, 0x0dde, 0x0135, 0x0789, 0x0ddd, 0xbbbc, 0xf346, 0x0467,
    0x0fff, 0x4eee, 0x3ddd, 0x0edd, 0x0dee, 0x0fff, 0x0fff, 0x0dee,
    0x0def, 0x08ab, 0x0fff, 0x7fff, 0xfabc, 0xf356, 0x0457, 0x0467,
    0x0fff, 0x0bcd, 0x4bde, 0x9bcc, 0x8dee, 0x8eff, 0x8fff, 0x9fff,
    0xadee, 0xeccd, 0xf689, 0xc357, 0x2356, 0x0356, 0x0467, 0x0467,
    0x0fff, 0x0ccd, 0x0bdd, 0x0cdd, 0x0aaa, 0x2234, 0x4135, 0x4346,
    0x5356, 0x2246, 0x0346, 0x0356, 0x0467, 0x0356, 0x0467, 0x0467,
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff,
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff,
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff,
    0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff, 0x0fff
  };
  surface = SDL_CreateSurface(16, 16, SDL_PIXELFORMAT_ARGB4444);

  // The icon is attached to the window pointer
  SDL_SetWindowIcon(window, surface);

  // ...and the surface containing the icon pixel data is no longer
      required.
  SDL_DestroySurface(surface);

  // Loop until the user closes the window or presses any key.
  for( ; e.type != SDL_EVENT_QUIT && e.type != SDL_EVENT_KEY_DOWN;
      SDL_PollEvent(&e));

  SDL_DestroyWindow(window); // Close and destroy the window.
  SDL_Quit();                // Clean up and exit.
  return 0;

}
```

Version

This function is available since SDL 3.0.0.

SDL_SetWindowInputFocus

Explicitly set input focus to the window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowInputFocus(SDL_Window *window);
```

Function Parameters

window	the window that should get the input focus

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

You almost certainly want SDL_RaiseWindow() instead of this function. Use this with caution, as you might give focus to a window that is completely obscured by other windows.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RaiseWindow

SDL_SetWindowKeyboardGrab

Set a window's keyboard grab mode.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowKeyboardGrab(SDL_Window *window, SDL_bool grabbed);
```

Function Parameters

window	The window for which the keyboard grab mode should be set.
grabbed	This is SDL_TRUE to grab keyboard, and SDL_FALSE to release.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Keyboard grab enables capture of system keyboard shortcuts like Alt+Tab or the Meta/Super key. Note that not all system keyboard shortcuts can be captured by applications (one example is Ctrl+Alt+Del on Windows).

This is primarily intended for specialized applications such as VNC clients or VM frontends. Normal games should not use keyboard grab.

When keyboard grab is enabled, SDL will continue to handle Alt+Tab when the window is full-screen to ensure the user is not trapped in your application. If you have a custom keyboard shortcut to exit fullscreen mode, you may suppress this behavior with `SDL_HINT_ALLOW_ALT_TAB_WHILE_GRABBED`.

If the caller enables a grab while another window is currently grabbed, the other window loses its grab in favor of the caller's window.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowKeyboardGrab
- SDL_SetWindowMouseGrab

SDL_SetWindowMaximumSize

Set the maximum size of a window's client area.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowMaximumSize(SDL_Window *window, int max_w, int max_h);
```

Function Parameters

window	the window to change
max_w	the maximum width of the window, or 0 for no limit
max_h	the maximum height of the window, or 0 for no limit

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowMaximumSize
- SDL_SetWindowMinimumSize

SDL_SetWindowMinimumSize

Set the minimum size of a window's client area.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowMinimumSize(SDL_Window *window, int min_w, int min_h);
```

Function Parameters

window	the window to change
min_w	the minimum width of the window, or 0 for no limit
min_h	the minimum height of the window, or 0 for no limit

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowMinimumSize
- SDL_SetWindowMaximumSize

SDL_SetWindowModalFor

Set the window as a modal to a parent window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowModalFor(SDL_Window *modal_window, SDL_Window
    *parent_window);
```

Function Parameters

modal_window	the window that should be set modal
parent_window	the parent window for the modal window

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If the window is already modal to an existing window, it will be reparented to the new owner. Setting the parent window to null unparents the modal window and removes modal status.

Setting a window as modal to a parent that is a descendent of the modal window results in undefined behavior.

Version

This function is available since SDL 3.0.0.

SDL_SetWindowMouseGrab

Set a window's mouse grab mode.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowMouseGrab(SDL_Window *window, SDL_bool grabbed);
```

Function Parameters

window	The window for which the mouse grab mode should be set.
grabbed	This is SDL_TRUE to grab mouse, and SDL_FALSE to release.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Mouse grab confines the mouse cursor to the window.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowMouseGrab
- SDL_SetWindowKeyboardGrab

SDL_SetWindowMouseRect

Confines the cursor to the specified area of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowMouseRect(SDL_Window *window, const SDL_Rect *rect);
```

Function Parameters

window	The window that will be associated with the barrier.
rect	A rectangle area in window-relative coordinates. If NULL the barrier for the specified window will be destroyed.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Note that this does NOT grab the cursor, it only defines the area a cursor is restricted to when the window has mouse focus.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowMouseRect
- SDL_SetWindowMouseGrab

SDL_SetWindowOpacity

Set the opacity for a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowOpacity(SDL_Window *window, float opacity);
```

Function Parameters

window	the window which will be made transparent or opaque
opacity	the opacity value (0.0f - transparent, 1.0f - opaque)

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The parameter `opacity` will be clamped internally between 0.0f (transparent) and 1.0f (opaque).

This function also returns -1 if setting the opacity isn't supported.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowOpacity

SDL_SetWindowPosition

Request that the window's position be set.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowPosition(SDL_Window *window, int x, int y);
```

Function Parameters

window	the window to reposition
x	the x coordinate of the window, or `SDL_WINDOWPOS_CENTERED` or `SDL_WINDOWPOS_UNDEFINED`
y	the y coordinate of the window, or `SDL_WINDOWPOS_CENTERED` or `SDL_WINDOWPOS_UNDEFINED`

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If, at the time of this request, the window is in a fixed-size state such as maximized, this request may be deferred until the window returns to a resizable state.

This can be used to reposition fullscreen-desktop windows onto a different display, however, exclusive fullscreen windows are locked to a specific display and can only be repositioned programmatically via SDL_SetWindowFullscreenMode().

On some windowing systems this request is asynchronous and the new coordinates may not have have been applied immediately upon the return of this function. If an immediate change is required, call SDL_SyncWindow() to block until the changes have taken effect.

When the window position changes, an SDL_EVENT_WINDOW_MOVED event will be emitted with the window's new coordinates. Note that the new coordinates may not match the exact coordinates requested, as some windowing systems can restrict the position of the window in certain scenarios (e.g. constraining the position so the window is always within desktop bounds). Additionally, as this is just a request, it can be denied by the windowing system.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowPosition
- SDL_SyncWindow

SDL_SetWindowResizable

Set the user-resizable state of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowResizable(SDL_Window *window, SDL_bool resizable);
```

Function Parameters

window	the window of which to change the resizable state
resizable	SDL_TRUE to allow resizing, SDL_FALSE to disallow

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This will add or remove the window's `SDL_WINDOW_RESIZABLE` flag and allow/disallow user resizing of the window. This is a no-op if the window's resizable state already matches the requested state.

You can't change the resizable state of a fullscreen window.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowFlags

SDL_SetWindowShape

Set the shape of a transparent window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowShape(SDL_Window *window, SDL_Surface *shape);
```

Function Parameters

window	the window
shape	the surface representing the shape of the window, or NULL to remove any current shape

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This sets the alpha channel of a transparent window and any fully transparent areas are also transparent to mouse clicks. If you are using something besides the SDL render API, then you are responsible for setting the alpha channel of the window yourself.

The shape is copied inside this function, so you can free it afterwards. If your shape surface changes, you should call SDL_SetWindowShape() again to update the window.

The window must have been created with the SDL_WINDOW_TRANSPARENT flag.

Version

This function is available since SDL 3.0.0.

SDL_SetWindowSize

Request that the size of a window's client area be set.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowSize(SDL_Window *window, int w, int h);
```

Function Parameters

window	the window to change
w	the width of the window, must be ¿ 0
h	the height of the window, must be ¿ 0

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

NULL can safely be passed as the `w` or `h` parameter if the width or height value is not desired.

If, at the time of this request, the window in a fixed-size state, such as maximized or fullscreen, the request will be deferred until the window exits this state and becomes resizable again.

To change the fullscreen mode of a window, use SDL_SetWindowFullscreenMode()

On some windowing systems, this request is asynchronous and the new window size may not have have been applied immediately upon the return of this function. If an immediate change is required, call SDL_SyncWindow() to block until the changes have taken effect.

When the window size changes, an SDL_EVENT_WINDOW_RESIZED event will be emitted with the new window dimensions. Note that the new dimensions may not match the exact size requested, as some windowing systems can restrict the window size in certain scenarios (e.g. constraining the size of the content area to remain within the usable desktop bounds). Additionally, as this is just a request, it can be denied by the windowing system.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowSize
- SDL_SetWindowFullscreenMode
- SDL_SyncWindow

SDL_SetWindowsMessageHook

Set a callback for every Windows message, run before TranslateMessage().

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_SetWindowsMessageHook(SDL_WindowsMessageHook callback, void
    *userdata);
```

Function Parameters

callback	The SDL_WindowsMessageHook function to call.
userdata	a pointer to pass to every iteration of `callback`

Remarks

The callback may modify the message, and should return SDL_TRUE if the message should continue to be processed, or SDL_FALSE to prevent further processing.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_WindowsMessageHook
- SDL_HINT_WINDOWS_ENABLE_MESSAGELOOP

SDL_SetWindowTitle

Set the title of a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SetWindowTitle(SDL_Window *window, const char *title);
```

Function Parameters

window	the window to change
title	the desired window title in UTF-8 format

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This string is expected to be in UTF-8 encoding.

Code Examples

```
// dynamically setting a window title

#include <SDL3/SDL.h>

int main(int argc, char* argv[]){

  SDL_Window *window;
  SDL_Event e;

  const char *titles[] = { // just for fun, let's make the title animate
      like a marquee and annoy users
    "t", "thi", "this w", "this win", "this windo", "this window's",
        "this window's ti", "this window's title",
    "chis window's title is", "chih window's title is ", "chih wandnw's
        title is ", "c h wandnw'g title is ",
    "c  h  a  nw'g titln is ", "c h a n  g  i  n ig ", "c h  a  n  g  i  n
        g!", "",
```

```
    "c h a n g i n g!", "", "c h a n g i n g!", "c h a n g i
        n g!"
  };

  SDL_Init(SDL_INIT_VIDEO); // Init SDL2

  // Create a window.
  window = SDL_CreateWindow(
    "This will surely be overwritten", 320, 240, SDL_WINDOW_RESIZABLE
  );

  // Enter the main loop. Press any key or hit the x to exit.
  for( ; e.type!=SDL_EVENT_QUIT&&e.type!=SDL_EVENT_KEY_DOWN;
      SDL_PollEvent(&e)){
    static int i = 0, t = 0;

    if(!(++t%9)){ // every 9th frame...
      SDL_SetWindowTitle(window, titles[i]);       // loop through the
      if(++i >= sizeof(titles)/sizeof(titles[0])) i = 0; // array of
          titles
    }

    SDL_Delay(10);

  }

  SDL_DestroyWindow(window);
  SDL_Quit();
  return 0;

}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowTitle

SDL_SetX11EventHook

Set a callback for every X11 event.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
void SDL_SetX11EventHook(SDL_X11EventHook callback, void *userdata);
```

Function Parameters

callback	The SDL_X11EventHook function to call.
userdata	a pointer to pass to every iteration of `callback`

Remarks

The callback may modify the event, and should return SDL_TRUE if the event should continue to be processed, or SDL_FALSE to prevent further processing.

Version

This function is available since SDL 3.0.0.

SDL_ShowCursor

Show the cursor.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
int SDL_ShowCursor(void);
```

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Code Examples

```
int main(int argc, char *argv[]) {
    /* ... */
    if (SDL_ShowCursor() < 0) {
      SDL_Log("Failed to show the cursor");
    }
    /* ... */
    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CursorVisible
- SDL_HideCursor

SDL_ShowMessageBox

Create a modal message box.

Header File

Defined in SDL3/SDL_messagebox.h

Syntax

```
int SDL_ShowMessageBox(const SDL_MessageBoxData *messageboxdata, int
    *buttonid);
```

Function Parameters

messageboxdata	the SDL_MessageBoxData structure with title, text and other options
buttonid	the pointer to which user id of hit button should be copied

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If your needs aren't complex, it might be easier to use SDL_ShowSimpleMessageBox.

This function should be called on the thread that created the parent window, or on the main thread if the messagebox has no parent. It will block execution of that thread until the user clicks a button or closes the messagebox.

This function may be called at any time, even before SDL_Init(). This makes it useful for reporting errors like a failure to create a renderer or OpenGL context.

On X11, SDL rolls its own dialog box with X11 primitives instead of a formal toolkit like GTK+ or Qt.

Note that if SDL_Init() would fail because there isn't any available video target, this function is likely to fail for the same reasons. If this is a concern, check the return value from this function and fall back to writing to stderr if you can.

Code Examples

```
#include <SDL3/SDL.h>

int main(int argc, char *argv[])
{
    const SDL_MessageBoxButtonData buttons[] = {
        { /* .flags, .buttonid, .text */    0, 0, "no" },
        { SDL_MESSAGEBOX_BUTTON_RETURNKEY_DEFAULT, 1, "yes" },
        { SDL_MESSAGEBOX_BUTTON_ESCAPEKEY_DEFAULT, 2, "cancel" },
    };
    const SDL_MessageBoxColorScheme colorScheme = {
        { /* .colors (.r, .g, .b) */
            /* [SDL_MESSAGEBOX_COLOR_BACKGROUND] */
            { 255,  0,  0 },
            /* [SDL_MESSAGEBOX_COLOR_TEXT] */
            {  0, 255,  0 },
            /* [SDL_MESSAGEBOX_COLOR_BUTTON_BORDER] */
            { 255, 255,  0 },
            /* [SDL_MESSAGEBOX_COLOR_BUTTON_BACKGROUND] */
            {  0,  0, 255 },
            /* [SDL_MESSAGEBOX_COLOR_BUTTON_SELECTED] */
            { 255,  0, 255 }
        }
    };
    const SDL_MessageBoxData messageboxdata = {
        SDL_MESSAGEBOX_INFORMATION, /* .flags */
        NULL, /* .window */
        "example message box", /* .title */
        "select a button", /* .message */
        SDL_arraysize(buttons), /* .numbuttons */
        buttons, /* .buttons */
        &colorScheme /* .colorScheme */
    };
    int buttonid;
    if (SDL_ShowMessageBox(&messageboxdata, &buttonid) < 0) {
        SDL_Log("error displaying message box");
        return 1;
    }
    if (buttonid == -1) {
        SDL_Log("no selection");
    } else {
        SDL_Log("selection was %s", buttons[buttonid].text);
    }
    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ShowSimpleMessageBox

SDL_ShowOpenFileDialog

Displays a dialog that lets the user select a file on their filesystem.

Header File

Defined in SDL3/SDL_dialog.h

Syntax

```
void SDL_ShowOpenFileDialog(SDL_DialogFileCallback callback, void
    *userdata, SDL_Window *window, const SDL_DialogFileFilter *filters,
    const char *default_location, SDL_bool allow_many);
```

Function Parameters

callback	An SDL_DialogFileCallback to be invoked when the user selects a file and accepts, or cancels the dialog, or an error occurs. The first argument is a null-terminated list of C strings, representing the paths chosen by the user. The list will be empty if the user canceled the dialog, and it will be NULL if an error occured. If an error occured, it can be fetched with SDL_GetError(). The second argument is the userdata pointer passed to the function. The third argument is the index of the filter selected by the user, or one past the index of the last filter (therefore the index of the terminating NULL filter) if no filter was chosen, or -1 if the platform does not support detecting the selected filter.
userdata	An optional pointer to pass extra data to the callback when it will be invoked.
window	The window that the dialog should be modal for. May be NULL. Not all platforms support this option.
filters	A null-terminated list of SDL_DialogFileFilter's. May be NULL. Not all platforms support this option, and platforms that do support it may allow the user to ignore the filters.
default_location	The default folder or file to start the dialog at. May be NULL. Not all platforms support this option.
allow_many	If non-zero, the user will be allowed to select multiple entries. Not all platforms support this option.

Remarks

This function should only be invoked from the main thread.

This is an asynchronous function; it will return immediately, and the result will be passed to the callback.

The callback will be invoked with a null-terminated list of files the user chose. The list will be empty if the user canceled the dialog, and it will be NULL if an error occured.

Note that the callback may be called from a different thread than the one the function was invoked on.

Depending on the platform, the user may be allowed to input paths that don't yet exist.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DialogFileCallback
- SDL_DialogFileFilter
- SDL_ShowSaveFileDialog
- SDL_ShowOpenFolderDialog

SDL_ShowOpenFolderDialog

Displays a dialog that lets the user select a folder on their filesystem.

Header File

Defined in SDL3/SDL_dialog.h

Syntax

```
void SDL_ShowOpenFolderDialog(SDL_DialogFileCallback callback, void
    *userdata, SDL_Window *window, const char *default_location,
    SDL_bool allow_many);
```

Function Parameters

callback	An SDL_DialogFileCallback to be invoked when the user selects a file and accepts, or cancels the dialog, or an error occurs. The first argument is a null-terminated list of C strings, representing the paths chosen by the user. The list will be empty if the user canceled the dialog, and it will be NULL if an error occured. If an error occured, it can be fetched with SDL_GetError(). The second argument is the userdata pointer passed to the function. The third argument is always -1 for SDL_ShowOpenFolderDialog.
userdata	An optional pointer to pass extra data to the callback when it will be invoked.
window	The window that the dialog should be modal for. May be NULL. Not all platforms support this option.
default_location	The default folder or file to start the dialog at. May be NULL. Not all platforms support this option.
allow_many	If non-zero, the user will be allowed to select multiple entries. Not all platforms support this option.

Remarks

This function should only be invoked from the main thread.

This is an asynchronous function; it will return immediately, and the result will be passed to the callback.

The callback will be invoked with a null-terminated list of files the user chose. The list will be empty if the user canceled the dialog, and it will be NULL if an error occured.

Note that the callback may be called from a different thread than the one the function was invoked on.

Depending on the platform, the user may be allowed to input paths that don't yet exist.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DialogFileCallback
- SDL_ShowOpenFileDialog
- SDL_ShowSaveFileDialog

SDL_ShowSaveFileDialog

Displays a dialog that lets the user choose a new or existing file on their filesystem.

Header File

Defined in SDL3/SDL_dialog.h

Syntax

```
void SDL_ShowSaveFileDialog(SDL_DialogFileCallback callback, void
    *userdata, SDL_Window *window, const SDL_DialogFileFilter *filters,
    const char *default_location);
```

Function Parameters

callback	An SDL_DialogFileCallback to be invoked when the user selects a file and accepts, or cancels the dialog, or an error occurs. The first argument is a null-terminated list of C strings, representing the paths chosen by the user. The list will be empty if the user canceled the dialog, and it will be NULL if an error occured. If an error occured, it can be fetched with SDL_GetError(). The second argument is the userdata pointer passed to the function. The third argument is the index of the filter selected by the user, or one past the index of the last filter (therefore the index of the terminating NULL filter) if no filter was chosen, or -1 if the platform does not support detecting the selected filter.
userdata	An optional pointer to pass extra data to the callback when it will be invoked.
window	The window that the dialog should be modal for. May be NULL. Not all platforms support this option.
filters	A null-terminated list of SDL_DialogFileFilter's. May be NULL. Not all platforms support this option, and platforms that do support it may allow the user to ignore the filters.
default_location	The default folder or file to start the dialog at. May be NULL. Not all platforms support this option.

Remarks

This function should only be invoked from the main thread.

This is an asynchronous function; it will return immediately, and the result will be passed to the callback.

The callback will be invoked with a null-terminated list of files the user chose. The list will be empty if the user canceled the dialog, and it will be NULL if an error occured.

Note that the callback may be called from a different thread than the one the function was invoked on.

The chosen file may or may not already exist.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_DialogFileCallback
- SDL_DialogFileFilter
- SDL_ShowOpenFileDialog
- SDL_ShowOpenFolderDialog

SDL_ShowSimpleMessageBox

Display a simple modal message box.

Header File

Defined in SDL3/SDL_messagebox.h

Syntax

```
int SDL_ShowSimpleMessageBox(Uint32 flags, const char *title, const char
    *message, SDL_Window *window);
```

Function Parameters

flags	an SDL_MessageBoxFlags value
title	UTF-8 title text
message	UTF-8 message text
window	the parent window, or NULL for no parent

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

If your needs aren't complex, this function is preferred over SDL_ShowMessageBox. `flags` may be any of the following:

- `SDL_MESSAGEBOX_ERROR`: error dialog
- `SDL_MESSAGEBOX_WARNING`: warning dialog
- `SDL_MESSAGEBOX_INFORMATION`: informational dialog

This function should be called on the thread that created the parent window, or on the main thread if the messagebox has no parent. It will block execution of that thread until the user clicks a button or closes the messagebox.

This function may be called at any time, even before SDL_Init(). This makes it useful for reporting errors like a failure to create a renderer or OpenGL context.

On X11, SDL rolls its own dialog box with X11 primitives instead of a formal toolkit like GTK+ or Qt.

Note that if SDL_Init() would fail because there isn't any available video target, this function is likely to fail for the same reasons. If this is a concern,

check the return value from this function and fall back to writing to stderr if you can.

Code Examples

```
SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_ERROR,
                         "Missing file",
                         "File is missing. Please reinstall the program.",
                         NULL);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_ShowMessageBox

SDL_ShowWindow

Show a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_ShowWindow(SDL_Window *window);
```

Function Parameters

window	the window to show

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Code Examples

```
SDL_Window *window;

SDL_ShowWindow(window);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_HideWindow
- SDL_RaiseWindow

SDL_ShowWindowSystemMenu

Display the system-level window menu.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_ShowWindowSystemMenu(SDL_Window *window, int x, int y);
```

Function Parameters

window	the window for which the menu will be displayed
x	the x coordinate of the menu, relative to the origin (top-left) of the client area
y	the y coordinate of the menu, relative to the origin (top-left) of the client area

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This default window menu is provided by the system and on some platforms provides functionality for setting or changing privileged state on the window, such as moving it between workspaces or displays, or toggling the always-on-top property.

On platforms or desktops where this is unsupported, this function does nothing.

Version

This function is available since SDL 3.0.0.

SDL_SignalCondition

Restart one of the threads that are waiting on the condition variable.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_SignalCondition(SDL_Condition *cond);
```

Function Parameters

cond	the condition variable to signal

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BroadcastCondition
- SDL_WaitCondition
- SDL_WaitConditionTimeout

SDL_SIMDGetAlignment

Report the alignment this system needs for SIMD allocations.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
size_t SDL_SIMDGetAlignment(void);
```

Return Value

Returns the alignment in bytes needed for available, known SIMD instructions.

Remarks

This will return the minimum number of bytes to which a pointer must be aligned to be compatible with SIMD instructions on the current machine. For example, if the machine supports SSE only, it will return 16, but if it supports AVX-512F, it'll return 64 (etc). This only reports values for instruction sets SDL knows about, so if your SDL build doesn't have SDL_HasAVX512F(), then it might return 16 for the SSE support it sees and not 64 for the AVX-512 instructions that exist but SDL doesn't know about. Plan accordingly.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_aligned_alloc
- SDL_aligned_free

SDL_sin

Compute the sine of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_sin(double x);
```

Function Parameters

x	floating point value, in radians

Return Value

Returns sine of `x`

Remarks

Domain: `-INF <= x <= INF`

Range: `-1 <= y <= 1`

This function operates on double-precision floating point values, use SDL_sinf for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_sinf
- SDL_asin
- SDL_cos

SDL_sinf

Compute the sine of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_sinf(float x);
```

Function Parameters

x	floating point value, in radians

Return Value

Returns sine of `x`

Remarks

Domain: `-INF <= x <= INF`

Range: `-1 <= y <= 1`

This function operates on single-precision floating point values, use SDL_sinf for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_sin
- SDL_asinf
- SDL_cosf

SDL_size_add_overflow

If a + b would overflow, return -1.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
SDL_FORCE_INLINE int SDL_size_add_overflow (size_t a, size_t b, size_t
    *ret);
```

Remarks

Otherwise store a + b via ret and return 0.

Version

This function is available since SDL 3.0.0.

SDL_size_mul_overflow

If a * b would overflow, return -1.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
SDL_FORCE_INLINE int SDL_size_mul_overflow (size_t a, size_t b, size_t
    *ret);
```

Remarks

Otherwise store a * b via ret and return 0.

Version

This function is available since SDL 3.0.0.

SDL_SoftStretch

Perform stretch blit between two surfaces of the same format.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
int SDL_SoftStretch(SDL_Surface *src, const SDL_Rect *srcrect,
    SDL_Surface *dst, const SDL_Rect *dstrect, SDL_ScaleMode scaleMode);
```

Function Parameters

src	the SDL_Surface structure to be copied from
srcrect	the SDL_Rect structure representing the rectangle to be copied
dst	the SDL_Surface structure that is the blit target
dstrect	the SDL_Rect structure representing the target rectangle in the destination surface
scaleMode	scale algorithm to be used

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Using SDL_SCALEMODE_NEAREST: fast, low quality. Using SDL_SCALEMODE_LINEAR: bilinear scaling, slower, better quality, only 32BPP.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BlitSurfaceScaled

SDL_sqrt

Compute the square root of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_sqrt(double x);
```

Function Parameters

x	floating point value. Must be greater than or equal to 0.

Return Value

Returns square root of `x`

Remarks

Domain: `0 <= x <= INF`

Range: `0 <= y <= INF`

This function operates on double-precision floating point values, use SDL_sqrtf for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_sqrtf

SDL_sqrtf

Compute the square root of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_sqrtf(float x);
```

Function Parameters

x	floating point value. Must be greater than or equal to 0.

Return Value

Returns square root of `x`

Remarks

Domain: `0 <= x <= INF`

Range: `0 <= y <= INF`

This function operates on single-precision floating point values, use SDL_sqrt for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_sqrt

SDL_StartTextInput

Start accepting Unicode text input events.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
void SDL_StartTextInput(void);
```

Remarks

This function will start accepting Unicode text input events in the focused SDL window, and start emitting SDL_TextInputEvent (SDL_EVENT_TEXT_INPUT) and SDL_TextEditingEvent (SDL_EVENT_TEXT_EDITING) events. Please use this function in pair with SDL_StopTextInput().

Text input events are not received by default.

On some platforms using this function activates the screen keyboard.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetTextInputRect
- SDL_StopTextInput

SDL_StopHapticEffect

Stop the haptic effect on its associated haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_StopHapticEffect(SDL_Haptic *haptic, int effect);
```

Function Parameters

haptic	the SDL_Haptic device to stop the effect on
effect	the ID of the haptic effect to stop

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RunHapticEffect
- SDL_StopHapticEffects

SDL_StopHapticEffects

Stop all the currently playing effects on a haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_StopHapticEffects(SDL_Haptic *haptic);
```

Function Parameters

haptic	the SDL_Haptic device to stop

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_RunHapticEffect
- SDL_StopHapticEffects

SDL_StopHapticRumble

Stop the simple rumble on a haptic device.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_StopHapticRumble(SDL_Haptic *haptic);
```

Function Parameters

haptic	the haptic device to stop the rumble effect on

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PlayHapticRumble

SDL_StopTextInput

Stop receiving any text input events.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
void SDL_StopTextInput(void);
```

Remarks

Text input events are not received by default.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StartTextInput

SDL_StorageReady

Checks if the storage container is ready to use.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
SDL_bool SDL_StorageReady(SDL_Storage *storage);
```

Function Parameters

storage	a storage container to query

Return Value

Returns SDL_TRUE if the container is ready, SDL_FALSE otherwise

Remarks

This function should be called in regular intervals until it returns SDL_TRUE - however, it is not recommended to spinwait on this call, as the backend may depend on a synchronous message loop.

Version

This function is available since SDL 3.0.0.

SDL_strcasecmp

Compare two null-terminated UTF-8 strings, case-insensitively.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_strcasecmp(const char *str1, const char *str2);
```

Function Parameters

str1	The first string to compare. NULL is not permitted!
str2	The second string to compare. NULL is not permitted!

Return Value

Returns less than zero if str1 is "less than" str2, greater than zero if str1 is "greater than" str2, and zero if the strings match exactly.

Remarks

This will work with Unicode strings, using a technique called "case-folding" to handle the vast majority of case-sensitive human languages regardless of system locale. It can deal with expanding values: a German Eszett character can compare against two ASCII 's' chars and be considered a match, for example. A notable exception: it does not handle the Turkish 'i' character; human language is complicated!

Since this handles Unicode, it expects the string to be well-formed UTF-8 and not a null-terminated string of arbitrary bytes. Bytes that are not valid UTF-8 are treated as Unicode character U+FFFD (REPLACEMENT CHARACTER), which is to say two strings of random bits may turn out to match if they convert to the same amount of replacement characters.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_strcmp

Compare two null-terminated UTF-8 strings.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_strcmp(const char *str1, const char *str2);
```

Function Parameters

str1	The first string to compare. NULL is not permitted!
str2	The second string to compare. NULL is not permitted!

Return Value

Returns less than zero if str1 is "less than" str2, greater than zero if str1 is "greater than" str2, and zero if the strings match exactly.

Remarks

Due to the nature of UTF-8 encoding, this will work with Unicode strings, since effectively this function just compares bytes until it hits a null-terminating character. Also due to the nature of UTF-8, this can be used with SDL_qsort() to put strings in (roughly) alphabetical order.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_strlwr

Convert a string to lowercase.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
char* SDL_strlwr(char *str);
```

Function Parameters

str	The string to convert in-place.

Return Value

Returns The `str` pointer passed into this function.

Remarks

WARNING: Regardless of system locale, this will only convert ASCII values 'A' through 'Z' to lowercase.

This function operates on a null-terminated string of bytes–even if it is malformed UTF-8!–and converts ASCII characters 'A' through 'Z' to their lowercase equivalents in-place, returning the original `str` pointer.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_strupr

SDL_strncasecmp

Compare two UTF-8 strings, case-insensitively, up to a number of bytes.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_strncasecmp(const char *str1, const char *str2, size_t maxlen);
```

Function Parameters

str1	The first string to compare. NULL is not permitted!
str2	The second string to compare. NULL is not permitted!
maxlen	The maximum number of bytes to compare.

Return Value

Returns less than zero if str1 is "less than" str2, greater than zero if str1 is "greater than" str2, and zero if the strings match exactly.

Remarks

This will work with Unicode strings, using a technique called "case-folding" to handle the vast majority of case-sensitive human languages regardless of system locale. It can deal with expanding values: a German Eszett character can compare against two ASCII 's' chars and be considered a match, for example. A notable exception: it does not handle the Turkish 'i' character; human language is complicated!

Since this handles Unicode, it expects the string to be well-formed UTF-8 and not a null-terminated string of arbitrary bytes. Bytes that are not valid UTF-8 are treated as Unicode character U+FFFD (REPLACEMENT CHARACTER), which is to say two strings of random bits may turn out to match if they convert to the same amount of replacement characters.

Note that while this function is intended to be used with UTF-8, `maxlen` specifies a _byte_ limit! If the limit lands in the middle of a multi-byte UTF-8 sequence, it may convert a portion of the final character to one or more Unicode character U+FFFD (REPLACEMENT CHARACTER) so as not to overflow a buffer. `maxlen` specifies a maximum number of bytes to compare; if the strings match to this number of bytes (or both have matched to a null-terminator character before this number of bytes), they will be considered equal.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_strncmp

Compare two UTF-8 strings up to a number of bytes.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_strncmp(const char *str1, const char *str2, size_t maxlen);
```

Function Parameters

str1	The first string to compare. NULL is not permitted!
str2	The second string to compare. NULL is not permitted!
maxlen	The maximum number of _bytes_ to compare.

Return Value

Returns less than zero if str1 is "less than" str2, greater than zero if str1 is "greater than" str2, and zero if the strings match exactly.

Remarks

Due to the nature of UTF-8 encoding, this will work with Unicode strings, since effectively this function just compares bytes until it hits a null-terminating character. Also due to the nature of UTF-8, this can be used with SDL_qsort() to put strings in (roughly) alphabetical order.

Note that while this function is intended to be used with UTF-8, it is doing a bytewise comparison, and `maxlen` specifies a _byte_ limit! If the limit lands in the middle of a multi-byte UTF-8 sequence, it will only compare a portion of the final character. `maxlen` specifies a maximum number of bytes to compare; if the strings match to this number of bytes (or both have matched to a null-terminator character before this number of bytes), they will be considered equal.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_strupr

Convert a string to uppercase.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
char* SDL_strupr(char *str);
```

Remarks

WARNING: Regardless of system locale, this will only convert ASCII values 'A' through 'Z' to uppercase.

This function operates on a null-terminated string of bytes–even if it is malformed UTF-8!–and converts ASCII characters 'a' through 'z' to their uppercase equivalents in-place, returning the original `str` pointer.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_strlwr

SDL_SurfaceHasColorKey

Returns whether the surface has a color key.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_bool SDL_SurfaceHasColorKey(SDL_Surface *surface);
```

Function Parameters

surface	the SDL_Surface structure to query

Return Value

Returns SDL_TRUE if the surface has a color key, SDL_FALSE otherwise.

Remarks

It is safe to pass a NULL `surface` here; it will return SDL_FALSE.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetSurfaceColorKey
- SDL_GetSurfaceColorKey

SDL_SurfaceHasRLE

Returns whether the surface is RLE enabled.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
SDL_bool SDL_SurfaceHasRLE(SDL_Surface *surface);
```

Function Parameters

surface	the SDL_Surface structure to query

Return Value

Returns SDL_TRUE if the surface is RLE enabled, SDL_FALSE otherwise.

Remarks

It is safe to pass a NULL `surface` here; it will return SDL_FALSE.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetSurfaceRLE

SDL_Swap16

Byte-swap an unsigned 16-bit number.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
SDL_FORCE_INLINE Uint16 SDL_Swap16(Uint16 x);
```

Function Parameters

x	the value to byte-swap.

Return Value

Returns `x`, with its bytes in the opposite endian order.

Remarks

This will always byte-swap the value, whether it's currently in the native byte-order of the system or not. You should use SDL_SwapLE16 or SDL_SwapBE16 instead, in most cases.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_Swap32

Byte-swap an unsigned 32-bit number.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
SDL_FORCE_INLINE Uint32 SDL_Swap32(Uint32 x);
```

Function Parameters

x	the value to byte-swap.

Return Value

Returns `x`, with its bytes in the opposite endian order.

Remarks

This will always byte-swap the value, whether it's currently in the native byte-order of the system or not. You should use SDL_SwapLE32 or SDL_SwapBE32 instead, in most cases.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_Swap64

Byte-swap an unsigned 64-bit number.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
SDL_FORCE_INLINE Uint32 SDL_Swap64(Uint64 x);
```

Function Parameters

x	the value to byte-swap.

Return Value

Returns `x`, with its bytes in the opposite endian order.

Remarks

This will always byte-swap the value, whether it's currently in the native byte-order of the system or not. You should use SDL_SwapLE64 or SDL_SwapBE64 instead, in most cases.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_SwapFloat

Byte-swap a floating point number.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
SDL_FORCE_INLINE float SDL_SwapFloat(float x);
```

Function Parameters

x	the value to byte-swap.

Return Value

Returns x, with its bytes in the opposite endian order.

Remarks

This will always byte-swap the value, whether it's currently in the native byte-order of the system or not. You should use SDL_SwapFloatLE or SDL_SwapFloatBE instead, in most cases.

Note that this is a forced-inline function in a header, and not a public API function available in the SDL library (which is to say, the code is embedded in the calling program and the linker and dynamic loader will not be able to find this function inside SDL itself).

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_SyncWindow

Block until any pending window state is finalized.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_SyncWindow(SDL_Window *window);
```

Function Parameters

window	the window for which to wait for the pending state to be applied

Return Value

Returns 0 on success, a positive value if the operation timed out before the window was in the requested state, or a negative error code on failure; call SDL_GetError() for more information.

Remarks

On asynchronous windowing systems, this acts as a synchronization barrier for pending window state. It will attempt to wait until any pending window state has been applied and is guaranteed to return within finite time. Note that for how long it can potentially block depends on the underlying window system, as window state changes may involve somewhat lengthy animations that must complete before the window is in its final requested state.

On windowing systems where changes are immediate, this does nothing.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetWindowSize
- SDL_SetWindowPosition
- SDL_SetWindowFullscreen
- SDL_MinimizeWindow

- SDL_MaximizeWindow
- SDL_RestoreWindow

SDL_tan

Compute the tangent of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_tan(double x);
```

Function Parameters

x	floating point value, in radians

Return Value

Returns tangent of `x`

Remarks

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`

This function operates on double-precision floating point values, use SDL_tanf for single-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_tanf
- SDL_sin
- SDL_cos
- SDL_atan
- SDL_atan2

SDL_tanf

Compute the tangent of `x`.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_tanf(float x);
```

Function Parameters

x	floating point value, in radians

Return Value

Returns tangent of `x`

Remarks

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`

This function operates on single-precision floating point values, use SDL_tanf for double-precision floats.

This function may use a different approximation across different versions, platforms and configurations. i.e, it can return a different value given the same input on different machines or operating systems, or if SDL is updated.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_tan
- SDL_sinf
- SDL_cosf
- SDL_atanf
- SDL_atan2f

SDL_TellIO

Determine the current read/write offset in an SDL_IOStream data stream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
Sint64 SDL_TellIO(SDL_IOStream *context);
```

Function Parameters

context	an SDL_IOStream data stream object from which to get the current offset

Return Value

Returns the current offset in the stream, or -1 if the information can not be determined.

Remarks

SDL_TellIO is actually a wrapper function that calls the SDL_IOStream's `seek` method, with an offset of 0 bytes from `SDL_IO_SEEK_CUR`, to simplify application development.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SeekIO

SDL_TextInputActive

Check whether or not Unicode text input events are enabled.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
SDL_bool SDL_TextInputActive(void);
```

Return Value

Returns SDL_TRUE if text input events are enabled else SDL_FALSE.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_StartTextInput

SDL_TimeFromWindows

Converts a Windows FILETIME (100-nanosecond intervals since January 1, 1601) to an SDL time.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
SDL_Time SDL_TimeFromWindows(Uint32 dwLowDateTime, Uint32
    dwHighDateTime);
```

Function Parameters

dwLowDateTime	the low portion of the Windows FILETIME value
dwHighDateTime	the high portion of the Windows FILETIME value

Return Value

Returns the converted SDL time

Remarks

This function takes the two 32-bit values of the FILETIME structure as parameters.

Version

This function is available since SDL 3.0.0.

SDL_TimeToDateTime

Converts an SDL_Time in nanoseconds since the epoch to a calendar time in the SDL_DateTime format.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
int SDL_TimeToDateTime(SDL_Time ticks, SDL_DateTime *dt, SDL_bool
    localTime);
```

Function Parameters

ticks	the SDL_Time to be converted
dt	the resulting SDL_DateTime
localTime	the resulting SDL_DateTime will be expressed in local time if true, otherwise it will be in Universal Coordinated Time (UTC)

Return Value

Returns 0 on success or -1 on error; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

SDL_TimeToWindows

Converts an SDL time into a Windows FILETIME (100-nanosecond intervals since January 1, 1601).

Header File

Defined in SDL3/SDL_time.h

Syntax

```
void SDL_TimeToWindows(SDL_Time ticks, Uint32 *dwLowDateTime, Uint32
    *dwHighDateTime);
```

Function Parameters

ticks	the time to convert
dwLowDateTime	a pointer filled in with the low portion of the Windows FILETIME value
dwHighDateTime	a pointer filled in with the high portion of the Windows FILETIME value

Remarks

This function fills in the two 32-bit values of the FILETIME structure.

Version

This function is available since SDL 3.0.0.

SDL_tolower

Convert low-ASCII English letters to lowercase.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_tolower(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns Lowercase version of x, or x if no conversion available.

Remarks

WARNING: Regardless of system locale, this will only convert ASCII values 'A' through 'Z' to lowercase.

This function returns the lowercase equivalent of `x`. If a character cannot be converted, or is already lowercase, this function returns `x`.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_toupper

Convert low-ASCII English letters to uppercase.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_toupper(int x);
```

Function Parameters

x	character value to check.

Return Value

Returns Capitalized version of x, or x if no conversion available.

Remarks

WARNING: Regardless of system locale, this will only convert ASCII values 'a' through 'z' to uppercase.

This function returns the uppercase equivalent of `x`. If a character cannot be converted, or is already uppercase, this function returns `x`.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_trunc

Truncate `x` to an integer.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
double SDL_trunc(double x);
```

Function Parameters

x	floating point value

Return Value

Returns `x` truncated to an integer

Remarks

Rounds `x` to the next closest integer to 0. This is equivalent to removing the fractional part of `x`, leaving only the integer part.

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`, y integer

This function operates on double-precision floating point values, use SDL_truncf for single-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_truncf
- SDL_fmod
- SDL_ceil
- SDL_floor
- SDL_round
- SDL_lround

SDL_truncf

Truncate `x` to an integer.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
float SDL_truncf(float x);
```

Function Parameters

x	floating point value

Return Value

Returns `x` truncated to an integer

Remarks

Rounds `x` to the next closest integer to 0. This is equivalent to removing the fractional part of `x`, leaving only the integer part.

Domain: `-INF <= x <= INF`

Range: `-INF <= y <= INF`, y integer

This function operates on single-precision floating point values, use SDL_truncf for double-precision floats.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_trunc
- SDL_fmodf
- SDL_ceilf
- SDL_floorf
- SDL_roundf
- SDL_lroundf

SDL_TryLockMutex

Try to lock a mutex without blocking.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_TryLockMutex(SDL_Mutex *mutex) SDL_TRY_ACQUIRE(0, mutex);
```

Function Parameters

mutex	the mutex to try to lock

Return Value

Returns 0 or `SDL_MUTEX_TIMEDOUT`

Remarks

This works just like SDL_LockMutex(), but if the mutex is not available, this function returns `SDL_MUTEX_TIMEDOUT` immediately.

This technique is useful if you need exclusive access to a resource but don't want to wait for it, and will return to it to try again later.

This function does not fail; if mutex is NULL, it will return 0 immediately having locked nothing. If the mutex is valid, this function will always either lock the mutex and return 0, or return SDL_MUTEX_TIMEOUT and lock nothing.

Code Examples

```
int status;
SDL_Mutex *mutex;

mutex = SDL_CreateMutex();
if (!mutex) {
  SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't create mutex\n");
  return 1;
}

status = SDL_TryLockMutex(mutex);

if (status == 0) {
  SDL_Log("Locked mutex\n");
```

```
  SDL_UnlockMutex(mutex);
} else if (status == SDL_MUTEX_TIMEDOUT) {
  /* Mutex not available for locking right now */
} else {
  SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "Couldn't lock mutex\n");
}

SDL_DestroyMutex(mutex);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockMutex
- SDL_UnlockMutex

SDL_TryLockRWLockForReading

Try to lock a read/write lock _for reading_ without blocking.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_TryLockRWLockForReading(SDL_RWLock *rwlock)
    SDL_TRY_ACQUIRE_SHARED(0, rwlock);
```

Function Parameters

rwlock	the rwlock to try to lock

Return Value

Returns 0 or `SDL_RWLOCK_TIMEDOUT`

Remarks

This works just like SDL_LockRWLockForReading(), but if the rwlock is not available, then this function returns `SDL_RWLOCK_TIMEDOUT` immediately.

This technique is useful if you need access to a resource but don't want to wait for it, and will return to it to try again later.

Trying to lock for read-only access can succeed if other threads are holding read-only locks, as this won't prevent access.

This function does not fail; if rwlock is NULL, it will return 0 immediately having locked nothing. If rwlock is valid, this function will always either lock the rwlock and return 0, or return SDL_RWLOCK_TIMEOUT and lock nothing.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockRWLockForReading
- SDL_TryLockRWLockForWriting
- SDL_UnlockRWLock

SDL_TryLockRWLockForWriting

Try to lock a read/write lock _for writing_ without blocking.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_TryLockRWLockForWriting(SDL_RWLock *rwlock) SDL_TRY_ACQUIRE(0,
    rwlock);
```

Function Parameters

rwlock	the rwlock to try to lock

Return Value

Returns 0 or `SDL_RWLOCK_TIMEDOUT`

Remarks

This works just like SDL_LockRWLockForWriting(), but if the rwlock is not available, this function returns `SDL_RWLOCK_TIMEDOUT` immediately.

This technique is useful if you need exclusive access to a resource but don't want to wait for it, and will return to it to try again later.

It is illegal for the owning thread to lock an already-locked rwlock for writing (read-only may be locked recursively, writing can not). Doing so results in undefined behavior.

It is illegal to request a write lock from a thread that already holds a read-only lock. Doing so results in undefined behavior. Unlock the read-only lock before requesting a write lock.

This function does not fail; if rwlock is NULL, it will return 0 immediately having locked nothing. If rwlock is valid, this function will always either lock the rwlock and return 0, or return SDL_RWLOCK_TIMEOUT and lock nothing.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockRWLockForWriting
- SDL_TryLockRWLockForReading
- SDL_UnlockRWLock

SDL_TryLockSpinlock

Try to lock a spin lock by setting it to a non-zero value.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
SDL_bool SDL_TryLockSpinlock(SDL_SpinLock *lock);
```

Function Parameters

lock	a pointer to a lock variable

Return Value

Returns SDL_TRUE if the lock succeeded, SDL_FALSE if the lock is already held.

Remarks

Please note that spinlocks are dangerous if you don't know what you're doing. Please be careful using any sort of spinlock!

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockSpinlock
- SDL_UnlockSpinlock

SDL_TryWaitSemaphore

See if a semaphore has a positive value and decrement it if it does.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_TryWaitSemaphore(SDL_Semaphore *sem);
```

Function Parameters

sem	the semaphore to wait on

Return Value

Returns 0 if the wait succeeds, `SDL_MUTEX_TIMEDOUT` if the wait would block, or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function checks to see if the semaphore pointed to by `sem` has a positive value and atomically decrements the semaphore value if it does. If the semaphore doesn't have a positive value, the function immediately returns SDL_MUTEX_TIMEDOUT.

Code Examples

```
// BEWARE: This code example was migrated from the SDL2 Wiki, by only
    updating the names.

void add_data_to_queue(void);
void get_data_from_queue(void);
int data_available(void);
void wait_for_threads(void);

SDL_AtomicInt done;
SDL_Semaphore *sem;
SDL_AtomicSet(&done, 0);
sem = SDL_CreateSemaphore(0);

Thread_A:
    while (!SDL_AtomicGet(&done)) {
```

```
        add_data_to_queue();
        SDL_PostSemaphore(sem);
    }
Thread_B:
    while (!SDL_AtomicGet(&done)) {
        if (SDL_TryWaitSemaphore(sem) == 0 && data_available()) {
            get_data_from_queue();
        }
        /* do other processing */
    }

SDL_AtomicSet(&done, 1);
SDL_PostSemaphore(sem);
wait_for_threads();
SDL_DestroySemaphore(sem);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PostSemaphore
- SDL_WaitSemaphore
- SDL_WaitSemaphoreTimeout

SDL_UnbindAudioStream

Unbind a single audio stream from its audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
void SDL_UnbindAudioStream(SDL_AudioStream *stream);
```

Function Parameters

stream	an audio stream to unbind from a device.

Remarks

This is a convenience function, equivalent to calling `SDL_UnbindAudioStreams(&stream, 1)`.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BindAudioStream

SDL_UnbindAudioStreams

Unbind a list of audio streams from their audio devices.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
void SDL_UnbindAudioStreams(SDL_AudioStream **streams, int num_streams);
```

Function Parameters

streams	an array of audio streams to unbind.
num_streams	Number streams listed in the `streams` array.

Remarks

The streams being unbound do not all have to be on the same device. All streams on the same device will be unbound atomically (data will stop flowing through all unbound streams on the same device at the same time).

Unbinding a stream that isn't bound to a device is a legal no-op.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BindAudioStreams

SDL_UnloadObject

Unload a shared object from memory.

Header File

Defined in SDL3/SDL_loadso.h

Syntax

```
void SDL_UnloadObject(void *handle);
```

Function Parameters

handle	a valid shared object handle returned by SDL_LoadObject()

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LoadObject

SDL_UnlockAudioStream

Unlock an audio stream for serialized access.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
int SDL_UnlockAudioStream(SDL_AudioStream *stream);
```

Function Parameters

stream	The audio stream to unlock.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This unlocks an audio stream after a call to SDL_LockAudioStream.

Thread Safety

You should only call this from the same thread that previously called SDL_LockAudioStream.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockAudioStream

SDL_UnlockJoysticks

Unlocking for atomic access to the joystick API.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
void SDL_UnlockJoysticks(void) SDL_RELEASE(SDL_joystick_lock);
```

Version

This function is available since SDL 3.0.0.

SDL_UnlockMutex

Unlock the mutex.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
void SDL_UnlockMutex(SDL_Mutex *mutex) SDL_RELEASE(mutex);
```

Function Parameters

mutex	the mutex to unlock.

Remarks

It is legal for the owning thread to lock an already-locked mutex. It must unlock it the same number of times before it is actually made available for other threads in the system (this is known as a "recursive mutex").

It is illegal to unlock a mutex that has not been locked by the current thread, and doing so results in undefined behavior.

Code Examples

¡¡Include(SDL_CreateMutex, , , from="## Begin Mutex Example", to="## End Mutex Example")¿¿

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockMutex
- SDL_TryLockMutex

SDL_UnlockProperties

Unlock a set of properties.

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
void SDL_UnlockProperties(SDL_PropertiesID props);
```

Function Parameters

props	the properties to unlock

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockProperties

SDL_UnlockRWLock

Unlock the read/write lock.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
void SDL_UnlockRWLock(SDL_RWLock *rwlock) SDL_RELEASE_GENERIC(rwlock);
```

Function Parameters

rwlock	the rwlock to unlock.

Remarks

Use this function to unlock the rwlock, whether it was locked for read-only or write operations.

It is legal for the owning thread to lock an already-locked read-only lock. It must unlock it the same number of times before it is actually made available for other threads in the system (this is known as a "recursive rwlock").

It is illegal to unlock a rwlock that has not been locked by the current thread, and doing so results in undefined behavior.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockRWLockForReading
- SDL_LockRWLockForWriting
- SDL_TryLockRWLockForReading
- SDL_TryLockRWLockForWriting

SDL_UnlockSpinlock

Unlock a spin lock by setting it to 0.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
void SDL_UnlockSpinlock(SDL_SpinLock *lock);
```

Function Parameters

lock	a pointer to a lock variable

Remarks

Always returns immediately. ***Please note that spinlocks are dangerous if you don't know what you're doing. Please be careful using any sort of spinlock!***

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockSpinlock
- SDL_TryLockSpinlock

SDL_UnlockSurface

Release a surface after directly accessing the pixels.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
void SDL_UnlockSurface(SDL_Surface *surface);
```

Function Parameters

surface	the SDL_Surface structure to be unlocked

Code Examples

¡¡Include(SDL_LockSurface, , , from="== Code Examples ==", to="== Remarks ==")¿¿

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockSurface

SDL_UnlockTexture

Unlock a texture, uploading the changes to video memory, if needed.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
void SDL_UnlockTexture(SDL_Texture *texture);
```

Function Parameters

texture	a texture locked by SDL_LockTexture()

Remarks

Warning: Please note that SDL_LockTexture() is intended to be write-only; it will not guarantee the previous contents of the texture will be provided. You must fully initialize any area of a texture that you lock before unlocking it, as the pixels might otherwise be uninitialized memory.

Which is to say: locking and immediately unlocking a texture can result in corrupted textures, depending on the renderer in use.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockTexture

SDL_UnregisterApp

Deregister the win32 window class from an SDL_RegisterApp call.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
void SDL_UnregisterApp(void);
```

Remarks

This can be called to undo the effects of SDL_RegisterApp.

Most applications do not need to, and should not, call this directly; SDL will call it when deinitializing the video subsystem.

It is safe to call this multiple times, as long as every call is eventually paired with a prior call to SDL_RegisterApp. The window class will only be deregistered when the registration counter in SDL_RegisterApp decrements to zero through calls to this function.

Version

This function is available since SDL 3.0.0.

SDL_UpdateGamepads

Manually pump gamepad updates if not using the loop.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
void SDL_UpdateGamepads(void);
```

Remarks

This function is called automatically by the event loop if events are enabled. Under such circumstances, it will not be necessary to call this function.

Version

This function is available since SDL 3.0.0.

SDL_UpdateHapticEffect

Update the properties of an effect.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
int SDL_UpdateHapticEffect(SDL_Haptic *haptic, int effect, const
    SDL_HapticEffect *data);
```

Function Parameters

haptic	the SDL_Haptic device that has the effect
effect	the identifier of the effect to update
data	an SDL_HapticEffect structure containing the new effect properties to use

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

Can be used dynamically, although behavior when dynamically changing direction may be strange. Specifically the effect may re-upload itself and start playing from the start. You also cannot change the type either when running SDL_UpdateHapticEffect().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateHapticEffect
- SDL_RunHapticEffect

SDL_UpdateJoysticks

Update the current state of the open joysticks.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
void SDL_UpdateJoysticks(void);
```

Remarks

This is called automatically by the event loop if any joystick events are enabled.

Version

This function is available since SDL 3.0.0.

SDL_UpdateNVTexture

Update a rectangle within a planar NV12 or NV21 texture with new pixels.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_UpdateNVTexture(SDL_Texture *texture,
                        const SDL_Rect *rect,
                        const Uint8 *Yplane, int Ypitch,
                        const Uint8 *UVplane, int UVpitch);
```

Function Parameters

texture	the texture to update
rect	a pointer to the rectangle of pixels to update, or NULL to update the entire texture.
Yplane	the raw pixel data for the Y plane.
Ypitch	the number of bytes between rows of pixel data for the Y plane.
UVplane	the raw pixel data for the UV plane.
UVpitch	the number of bytes between rows of pixel data for the UV plane.

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

You can use SDL_UpdateTexture() as long as your pixel data is a contiguous block of NV12/21 planes in the proper order, but this function is available if your pixel data is not contiguous.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_UpdateTexture
- SDL_UpdateYUVTexture

SDL_UpdateSensors

Update the current state of the open sensors.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
void SDL_UpdateSensors(void);
```

Remarks

This is called automatically by the event loop if sensor events are enabled.

This needs to be called from the thread that initialized the sensor subsystem.

Version

This function is available since SDL 3.0.0.

SDL_UpdateTexture

Update the given texture rectangle with new pixel data.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
int SDL_UpdateTexture(SDL_Texture *texture, const SDL_Rect *rect, const
    void *pixels, int pitch);
```

Function Parameters

texture	the texture to update
rect	an SDL_Rect structure representing the area to update, or NULL to update the entire texture
pixels	the raw pixel data in the format of the texture
pitch	the number of bytes in a row of pixel data, including padding between lines

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

The pixel data must be in the pixel format of the texture. Use SDL_QueryTexture() to query the pixel format of the texture.

This is a fairly slow function, intended for use with static textures that do not change often.

If the texture is intended to be updated often, it is preferred to create the texture as streaming and use the locking functions referenced below. While this function will work with streaming textures, for optimization reasons you may not get the pixels back if you lock the texture afterward.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_LockTexture
- SDL_UnlockTexture
- SDL_UpdateNVTexture
- SDL_UpdateYUVTexture

SDL_UpdateWindowSurface

Copy the window surface to the screen.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_UpdateWindowSurface(SDL_Window *window);
```

Function Parameters

window	the window to update

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This is the function you use to reflect any changes to the surface on the screen.

This function is equivalent to the SDL 1.2 API SDL_Flip().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowSurface
- SDL_UpdateWindowSurfaceRects

SDL_UpdateWindowSurfaceRects

Copy areas of the window surface to the screen.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
int SDL_UpdateWindowSurfaceRects(SDL_Window *window, const SDL_Rect
    *rects, int numrects);
```

Function Parameters

window	the window to update
rects	an array of SDL_Rect structures representing areas of the surface to copy, in pixels
numrects	the number of rectangles

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This is the function you use to reflect changes to portions of the surface on the screen.

This function is equivalent to the SDL 1.2 API SDL_UpdateRects().

Note that this function will update _at least_ the rectangles specified, but this is only intended as an optimization; in practice, this might update more of the screen (or all of the screen!), depending on what method SDL uses to send pixels to the system.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowSurface
- SDL_UpdateWindowSurface

SDL_UpdateYUVTexture

Update a rectangle within a planar YV12 or IYUV texture with new pixel data.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
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);
```

Function Parameters

texture	the texture to update
rect	a pointer to the rectangle of pixels to update, or NULL to update the entire texture
Yplane	the raw pixel data for the Y plane
Ypitch	the number of bytes between rows of pixel data for the Y plane
Uplane	the raw pixel data for the U plane
Upitch	the number of bytes between rows of pixel data for the U plane
Vplane	the raw pixel data for the V plane
Vpitch	the number of bytes between rows of pixel data for the V plane

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

You can use SDL_UpdateTexture() as long as your pixel data is a contiguous block of Y and U/V planes in the proper order, but this function is available if your pixel data is not contiguous.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_UpdateNVTexture
- SDL_UpdateTexture

SDL_Vulkan_CreateSurface

Create a Vulkan rendering surface for a window.

Header File

Defined in SDL3/SDL_vulkan.h

Syntax

```
SDL_bool SDL_Vulkan_CreateSurface(SDL_Window *window,
                                  VkInstance instance,
                                  const struct VkAllocationCallbacks
                                      *allocator,
                                  VkSurfaceKHR* surface);
```

Function Parameters

window	The window to which to attach the Vulkan surface
instance	The Vulkan instance handle
allocator	A VkAllocationCallbacks struct, which lets the app set the allocator that creates the surface. Can be NULL.
surface	A pointer to a VkSurfaceKHR handle to output the newly created surface

Return Value

Returns SDL_TRUE on success, SDL_FALSE on error.

Remarks

The `window` must have been created with the `SDL_WINDOW_VULKAN` flag and `instance` must have been created with extensions returned by SDL_Vulkan_GetInstanceExtensions() enabled.

If `allocator` is NULL, Vulkan will use the system default allocator. This argument is passed directly to Vulkan and isn't used by SDL itself.

Code Examples

```
extern void handle_error(void);

SDL_Window *window;
VkInstance instance;
VkSurfaceKHR surface;
```

```
if (!SDL_Vulkan_CreateSurface(window, instance, NULL, &surface))
    handle_error();
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Vulkan_GetInstanceExtensions

SDL_Vulkan_GetInstanceExtensions

Get the Vulkan instance extensions needed for vkCreateInstance.

Header File

Defined in SDL3/SDL_vulkan.h

Syntax

```
char const* const* SDL_Vulkan_GetInstanceExtensions(Uint32 *count);
```

Function Parameters

count	a pointer filled in with the number of extensions returned.

Return Value

Returns An array of extension name strings on success, NULL on error.

Remarks

This should be called after either calling SDL_Vulkan_LoadLibrary() or creating an SDL_Window with the `SDL_WINDOW_VULKAN` flag.

On return, the variable pointed to by `count` will be set to the number of elements returned, suitable for using with VkInstanceCreateInfo::enabledExtensionCount, and the returned array can be used with VkInstanceCreateInfo::ppEnabledExtensionNames, for calling Vulkan's vkCreateInstance API.

You should not free the returned array; it is owned by SDL.

Code Examples

```
extern void handle_error(void);
#ifndef VK_EXT_DEBUG_REPORT_EXTENSION_NAME
#define VK_EXT_DEBUG_REPORT_EXTENSION_NAME "VK_EXT_debug_report"
#endif

int count_instance_extensions;
const char * const *instance_extensions =
    SDL_Vulkan_GetInstanceExtensions(&count_instance_extensions);

if (instance_extensions == NULL) { handle_error(); }

Uint32 count_extensions = count_instance_extensions;
```

```
const char **extensions = SDL_malloc(count_extensions * sizeof(const
    char *));
extensions[0] = VK_EXT_DEBUG_REPORT_EXTENSION_NAME;
SDL_memcpy(&extensions[1], instance_extensions,
    count_instance_extensions * sizeof(const char*));

// Now we can make the Vulkan instance
VkInstanceCreateInfo create_info = {};
create_info.enabledExtensionCount = count_extensions;
create_info.ppEnabledExtensionNames = extensions;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, NULL, &instance);
SDL_free(extensions);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Vulkan_CreateSurface

SDL_Vulkan_GetVkGetInstanceProcAddr

Get the address of the `vkGetInstanceProcAddr` function.

Header File

Defined in SDL3/SDL_vulkan.h

Syntax

```
SDL_FunctionPointer SDL_Vulkan_GetVkGetInstanceProcAddr(void);
```

Return Value

Returns the function pointer for `vkGetInstanceProcAddr` or NULL on error.

Remarks

This should be called after either calling SDL_Vulkan_LoadLibrary() or creating an SDL_Window with the `SDL_WINDOW_VULKAN` flag.

The actual type of the returned function pointer is PFN_vkGetInstanceProcAddr, but that isn't available because the Vulkan headers are not included here. You should cast the return value of this function to that type, e.g. `vkGetInstanceProcAddr = (PFN_vkGetInstanceProcAddr)SDL_Vulkan_GetVkGetInstanceProcAddr();`

Version

This function is available since SDL 3.0.0.

SDL_Vulkan_LoadLibrary

Dynamically load the Vulkan loader library.

Header File

Defined in SDL3/SDL_vulkan.h

Syntax

```
int SDL_Vulkan_LoadLibrary(const char *path);
```

Function Parameters

path	The platform dependent Vulkan loader library name or NULL

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This should be called after initializing the video driver, but before creating any Vulkan windows. If no Vulkan loader library is loaded, the default library will be loaded upon creation of the first Vulkan window.

It is fairly common for Vulkan applications to link with libvulkan instead of explicitly loading it at run time. This will work with SDL provided the application links to a dynamic library and both it and SDL use the same search path.

If you specify a non-NULL `path`, an application should retrieve all of the Vulkan functions it uses from the dynamic library using SDL_Vulkan_GetVkGetInstanceProcAddr unless you can guarantee `path` points to the same vulkan loader library the application linked to.

On Apple devices, if `path` is NULL, SDL will attempt to find the `vkGetInstanceProcAddr` address within all the Mach-O images of the current process. This is because it is fairly common for Vulkan applications to link with libvulkan (and historically MoltenVK was provided as a static library). If it is not found, on macOS, SDL will attempt to load `vulkan.framework/vulkan`, `libvulkan.1.dylib`, `MoltenVK.framework/MoltenVK`, and `libMoltenVK.dylib`, in that order. On iOS, SDL will attempt to load `libMoltenVK.dylib`. Applications using a dynamic framework or .dylib must ensure it is included in its application bundle.

On non-Apple devices, application linking with a static libvulkan is not supported. Either do not link to the Vulkan loader or link to a dynamic library version.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Vulkan_GetVkInstanceProcAddr
- SDL_Vulkan_UnloadLibrary

SDL_Vulkan_UnloadLibrary

Unload the Vulkan library previously loaded by SDL_Vulkan_LoadLibrary().

Header File

Defined in SDL3/SDL_vulkan.h

Syntax

```
void SDL_Vulkan_UnloadLibrary(void);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Vulkan_LoadLibrary

SDL_WaitCondition

Wait until a condition variable is signaled.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_WaitCondition(SDL_Condition *cond, SDL_Mutex *mutex);
```

Function Parameters

cond	the condition variable to wait on
mutex	the mutex used to coordinate thread access

Return Value

Returns 0 when it is signaled or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function unlocks the specified `mutex` and waits for another thread to call SDL_SignalCondition() or SDL_BroadcastCondition() on the condition variable `cond`. Once the condition variable is signaled, the mutex is re-locked and the function returns.

The mutex must be locked before calling this function. Locking the mutex recursively (more than once) is not supported and leads to undefined behavior.

This function is the equivalent of calling SDL_WaitConditionTimeout() with a time length of -1.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BroadcastCondition
- SDL_SignalCondition
- SDL_WaitConditionTimeout

SDL_WaitConditionTimeout

Wait until a condition variable is signaled or a certain time has passed.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_WaitConditionTimeout(SDL_Condition *cond,
                     SDL_Mutex *mutex, Sint32 timeoutMS);
```

Function Parameters

cond	the condition variable to wait on
mutex	the mutex used to coordinate thread access
timeoutMS	the maximum time to wait, in milliseconds, or -1 to wait indefinitely

Return Value

Returns 0 if the condition variable is signaled, `SDL_MUTEX_TIMEDOUT` if the condition is not signaled in the allotted time, or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function unlocks the specified `mutex` and waits for another thread to call SDL_SignalCondition() or SDL_BroadcastCondition() on the condition variable `cond`, or for the specified time to elapse. Once the condition variable is signaled or the time elapsed, the mutex is re-locked and the function returns.

The mutex must be locked before calling this function. Locking the mutex recursively (more than once) is not supported and leads to undefined behavior.

Code Examples

```
SDL_bool condition = SDL_FALSE;
SDL_Mutex *lock;
SDL_Condition *cond;
lock = SDL_CreateMutex();
cond = SDL_CreateCondition();

Thread_A:
    const Uint32 timeout = 1000; /* wake up every second */
```

```
    SDL_bool done = SDL_FALSE;
    while (!done) {
        SDL_LockMutex(lock);
        while (!condition && SDL_WaitConditionTimeout(cond, lock,
            timeout) == 0) {
            continue;
        }
        SDL_UnlockMutex(lock);
        if (condition) {
            /* ... */
        }
        /* ... do some periodic work */
    }
Thread_B:
    SDL_LockMutex(lock);
    /* ... */
    condition = SDL_TRUE;
    /* ... */
    SDL_SignalCondition(cond);
    SDL_UnlockMutex(lock);

SDL_DestroyCondition(cond);
SDL_DestroyMutex(lock);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_BroadcastCondition
- SDL_SignalCondition
- SDL_WaitCondition

SDL_WaitEvent

Wait indefinitely for the next available event.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
SDL_bool SDL_WaitEvent(SDL_Event *event);
```

Function Parameters

event	the SDL_Event structure to be filled in with the next event from the queue, or NULL

Return Value

Returns SDL_TRUE on success or SDL_FALSE if there was an error while waiting for events; call SDL_GetError() for more information.

Remarks

If `event` is not NULL, the next event is removed from the queue and stored in the SDL_Event structure pointed to by `event`.

As this function may implicitly call SDL_PumpEvents(), you can only call this function in the thread that initialized the video subsystem.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PollEvent
- SDL_PushEvent
- SDL_WaitEventTimeout

SDL_WaitEventTimeout

Wait until the specified timeout (in milliseconds) for the next available event.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
SDL_bool SDL_WaitEventTimeout(SDL_Event *event, Sint32 timeoutMS);
```

Function Parameters

event	the SDL_Event structure to be filled in with the next event from the queue, or NULL
timeoutMS	the maximum number of milliseconds to wait for the next available event

Return Value

Returns SDL_TRUE if this got an event or SDL_FALSE if the timeout elapsed without any events available.

Remarks

If `event` is not NULL, the next event is removed from the queue and stored in the SDL_Event structure pointed to by `event`.

As this function may implicitly call SDL_PumpEvents(), you can only call this function in the thread that initialized the video subsystem.

The timeout is not guaranteed, the actual wait time could be longer due to system scheduling.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PollEvent
- SDL_PushEvent
- SDL_WaitEvent

SDL_WaitSemaphore

Wait until a semaphore has a positive value and then decrements it.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_WaitSemaphore(SDL_Semaphore *sem);
```

Function Parameters

sem	the semaphore wait on

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function suspends the calling thread until either the semaphore pointed to by `sem` has a positive value or the call is interrupted by a signal or error. If the call is successful it will atomically decrement the semaphore value.

This function is the equivalent of calling SDL_WaitSemaphoreTimeout() with a time length of -1.

Code Examples

```
// BEWARE: This code example was migrated from the SDL2 Wiki, by only
    updating the names.

#define NB_WAITER 10
SDL_Semaphore *sem;
// Increments the semaphore every 2s
int poster_thread() {
  for (int i = 0; i < NB_WAITER; i++) {
    SDL_PostSemaphore(sem);
    SDL_Delay(2 * 1000);
  }
  return 0;
}
int waiter_thread() {
```

```
  int status;
  status = SDL_WaitSemaphore(sem);

  if (status == 0) {
    SDL_Log("Semaphore was decremented.\n");
  } else {
    SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "An error has occured
        while waiting: %s\n", SDL_GetError());
  }
  return 0;
}
int main() {
  sem = SDL_CreateSemaphore(0);
  create_and_wait_threads(); // 1 poster, 10 waiters
  SDL_DestroySemaphore(sem);
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PostSemaphore
- SDL_TryWaitSemaphore
- SDL_WaitSemaphoreTimeout

SDL_WaitSemaphoreTimeout

Wait until a semaphore has a positive value and then decrements it.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
int SDL_WaitSemaphoreTimeout(SDL_Semaphore *sem, Sint32 timeoutMS);
```

Function Parameters

sem	the semaphore to wait on
timeoutMS	the length of the timeout, in milliseconds

Return Value

Returns 0 if the wait succeeds, `SDL_MUTEX_TIMEDOUT` if the wait does not succeed in the allotted time, or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function suspends the calling thread until either the semaphore pointed to by `sem` has a positive value, the call is interrupted by a signal or error, or the specified time has elapsed. If the call is successful it will atomically decrement the semaphore value.

Code Examples

```
// BEWARE: This code example was migrated from the SDL2 Wiki, by only
    updating the names.

void add_data_to_queue(void);
void get_data_from_queue(void);
int data_available(void);
void wait_for_threads(void);

SDL_AtomicInt done;
SDL_Semaphore *sem;
SDL_AtomicSet(&done, 0);
sem = SDL_CreateSemaphore(0);
```

```
Thread_A:
    while (!SDL_AtomicGet(&done)) {
        add_data_to_queue();
        SDL_PostSemaphore(sem);
    }
Thread_B:
    const Uint32 timeout = 1000; /* wake up every second */
    while (!SDL_AtomicGet(&done)) {
        if (SDL_WaitSemaphoreTimeout(sem, timeout) == 0 &&
            data_available()) {
            get_data_from_queue();
        }
        /* ... do other processing */
    }

SDL_AtomicSet(&done, 1);
SDL_PostSemaphore(sem);
wait_for_threads();
SDL_DestroySemaphore(sem);
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_PostSemaphore
- SDL_TryWaitSemaphore
- SDL_WaitSemaphore

SDL_WaitThread

Wait for a thread to finish.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
void SDL_WaitThread(SDL_Thread * thread, int *status);
```

Function Parameters

thread	the SDL_Thread pointer that was returned from the SDL_CreateThread() call that started this thread
status	pointer to an integer that will receive the value returned from the thread function by its 'return', or NULL to not receive such value back.

Remarks

Threads that haven't been detached will remain (as a "zombie") until this function cleans them up. Not doing so is a resource leak.

Once a thread has been cleaned up through this function, the SDL_Thread that references it becomes invalid and should not be referenced again. As such, only one thread may call SDL_WaitThread() on another.

The return code for the thread function is placed in the area pointed to by `status`, if `status` is not NULL.

You may not wait on a thread that has been used in a call to SDL_DetachThread(). Use either that function or this one, but not both, or behavior is undefined.

It is safe to pass a NULL thread to this function; it is a no-op.

Note that the thread pointer is freed by this function and is not valid afterward.

Code Examples

```
#include <SDL3/SDL.h>

// Very simple thread - counts 0 to 9 delaying 50ms between increments
static int TestThread(void *ptr)
{
    int cnt;

    for (cnt = 0; cnt < 10; ++cnt) {
```

```
        SDL_Log("Thread counter: %d\n", cnt);
        SDL_Delay(50);
    }

    // Return the final value to the SDL_WaitThread function above
    return cnt;
}

int main(int argc, char *argv[])
{
    SDL_Thread *thread;
    int        threadReturnValue;

    SDL_Log("Simple SDL_CreateThread test:");

    // Simply create a thread
    thread = SDL_CreateThread(TestThread, "TestThread", (void *)NULL);

    if (NULL == thread) {
        SDL_LogError(SDL_LOG_CATEGORY_APPLICATION, "SDL_CreateThread
            failed: %s\n", SDL_GetError());
    } else {
        // Wait for the thread to complete. The thread functions return
            code will
        //      be placed in the "threadReturnValue" variable when it
            completes.
        //
        SDL_WaitThread(thread, &threadReturnValue);
        SDL_Log("Thread returned value: %d\n", threadReturnValue);
    }

    return 0;
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_CreateThread
- SDL_DetachThread

SDL_WarpMouseGlobal

Move the mouse to the given position in global screen space.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
int SDL_WarpMouseGlobal(float x, float y);
```

Function Parameters

x	the x coordinate
y	the y coordinate

Return Value

Returns 0 on success or a negative error code on failure; call SDL_GetError() for more information.

Remarks

This function generates a mouse motion event.

A failure of this function usually means that it is unsupported by a platform.

Note that this function will appear to succeed, but not actually move the mouse when used over Microsoft Remote Desktop.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_WarpMouseInWindow

SDL_WarpMouseInWindow

Move the mouse cursor to the given position within the window.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
void SDL_WarpMouseInWindow(SDL_Window * window,
                           float x, float y);
```

Function Parameters

window	the window to move the mouse into, or NULL for the current mouse focus
x	the x coordinate within the window
y	the y coordinate within the window

Remarks

This function generates a mouse motion event if relative mode is not enabled. If relative mode is enabled, you can force mouse events for the warp by setting the SDL_HINT_MOUSE_RELATIVE_WARP_MOTION hint.

Note that this function will appear to succeed, but not actually move the mouse when used over Microsoft Remote Desktop.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_WarpMouseGlobal

SDL_WasInit

Get a mask of the specified subsystems which are currently initialized.

Header File

Defined in SDL3/SDL_init.h

Syntax

```
Uint32 SDL_WasInit(Uint32 flags);
```

Function Parameters

flags	any of the flags used by SDL_Init(); see SDL_Init for details.

Return Value

Returns a mask of all initialized subsystems if `flags` is 0, otherwise it returns the initialization status of the specified subsystems.

Code Examples

```
/* Get init data on all the subsystems */
Uint32 subsystem_init;

subsystem_init = SDL_WasInit(SDL_INIT_AUDIO | SDL_INIT_VIDEO);

if (subsystem_init & SDL_INIT_VIDEO) {
    SDL_Log("Video is initialized.");
} else {
    SDL_Log("Video is not initialized.");
}
```

```
/* Just check for one specific subsystem */

if (SDL_WasInit(SDL_INIT_VIDEO) != 0) {
    SDL_Log("Video is initialized.");
} else {
    SDL_Log("Video is not initialized.");
}
```

```
/* Check for two subsystems */
Uint32 subsystem_mask = SDL_INIT_VIDEO | SDL_INIT_AUDIO;
```

```
if (SDL_WasInit(subsystem_mask) == subsystem_mask) {
    SDL_Log("Video and Audio initialized.");
} else {
    SDL_Log("Video and Audio not initialized.");
}
```

Version

This function is available since SDL 3.0.0.

See Also

- SDL_Init
- SDL_InitSubSystem

SDL_wcscasecmp

Compare two null-terminated wide strings, case-insensitively.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_wcscasecmp(const wchar_t *str1, const wchar_t *str2);
```

Function Parameters

str1	The first string to compare. NULL is not permitted!
str2	The second string to compare. NULL is not permitted!

Return Value

Returns less than zero if str1 is "less than" str2, greater than zero if str1 is "greater than" str2, and zero if the strings match exactly.

Remarks

This will work with Unicode strings, using a technique called "case-folding" to handle the vast majority of case-sensitive human languages regardless of system locale. It can deal with expanding values: a German Eszett character can compare against two ASCII 's' chars and be considered a match, for example. A notable exception: it does not handle the Turkish 'i' character; human language is complicated!

Depending on your platform, "wchar_t" might be 2 bytes, and expected to be UTF-16 encoded (like Windows), or 4 bytes in UTF-32 format. Since this handles Unicode, it expects the string to be well-formed and not a null-terminated string of arbitrary bytes. Characters that are not valid UTF-16 (or UTF-32) are treated as Unicode character U+FFFD (REPLACEMENT CHARACTER), which is to say two strings of random bits may turn out to match if they convert to the same amount of replacement characters.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_wcscmp

Compare two null-terminated wide strings.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_wcscmp(const wchar_t *str1, const wchar_t *str2);
```

Function Parameters

str1	The first string to compare. NULL is not permitted!
str2	The second string to compare. NULL is not permitted!

Return Value

Returns less than zero if str1 is "less than" str2, greater than zero if str1 is "greater than" str2, and zero if the strings match exactly.

Remarks

This only compares wchar_t values until it hits a null-terminating character; it does not care if the string is well-formed UTF-16 (or UTF-32, depending on your platform's wchar_t size), or uses valid Unicode values.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_wcsncasecmp

Compare two wide strings, case-insensitively, up to a number of wchar_t.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_wcsncasecmp(const wchar_t *str1, const wchar_t *str2, size_t
    maxlen);
```

Function Parameters

str1	The first string to compare. NULL is not permitted!
str2	The second string to compare. NULL is not permitted!
maxlen	The maximum number of wchar_t values to compare.

Return Value

Returns less than zero if str1 is "less than" str2, greater than zero if str1 is "greater than" str2, and zero if the strings match exactly.

Remarks

This will work with Unicode strings, using a technique called "case-folding" to handle the vast majority of case-sensitive human languages regardless of system locale. It can deal with expanding values: a German Eszett character can compare against two ASCII 's' chars and be considered a match, for example. A notable exception: it does not handle the Turkish 'i' character; human language is complicated!

Depending on your platform, "wchar_t" might be 2 bytes, and expected to be UTF-16 encoded (like Windows), or 4 bytes in UTF-32 format. Since this handles Unicode, it expects the string to be well-formed and not a null-terminated string of arbitrary bytes. Characters that are not valid UTF-16 (or UTF-32) are treated as Unicode character U+FFFD (REPLACEMENT CHARACTER), which is to say two strings of random bits may turn out to match if they convert to the same amount of replacement characters.

Note that while this function might deal with variable-sized characters, `maxlen` specifies a _wchar_ limit! If the limit lands in the middle of a multi-byte UTF-16 sequence, it may convert a portion of the final character to one or more Unicode character U+FFFD (REPLACEMENT CHARACTER) so as not to overflow a buffer. `maxlen` specifies a maximum number of wchar_t values to compare; if the strings match to this number of wchar_t (or both have

matched to a null-terminator character before this number of bytes), they will be considered equal.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_wcsncmp

Compare two wide strings up to a number of wchar_t values.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
int SDL_wcsncmp(const wchar_t *str1, const wchar_t *str2, size_t maxlen);
```

Function Parameters

str1	The first string to compare. NULL is not permitted!
str2	The second string to compare. NULL is not permitted!
maxlen	The maximum number of wchar_t to compare.

Return Value

Returns less than zero if str1 is "less than" str2, greater than zero if str1 is "greater than" str2, and zero if the strings match exactly.

Remarks

This only compares wchar_t values; it does not care if the string is well-formed UTF-16 (or UTF-32, depending on your platform's wchar_t size), or uses valid Unicode values.

Note that while this function is intended to be used with UTF-16 (or UTF-32, depending on your platform's definition of wchar_t), it is comparing raw wchar_t values and not Unicode codepoints: `maxlen` specifies a wchar_t limit! If the limit lands in the middle of a multi-wchar UTF-16 sequence, it will only compare a portion of the final character. `maxlen` specifies a maximum number of wchar_t to compare; if the strings match to this number of wide chars (or both have matched to a null-terminator character before this count), they will be considered equal.

Thread Safety

It is safe to call this function from any thread.

Version

This function is available since SDL 3.0.0.

SDL_WindowHasSurface

Return whether the window has a surface associated with it.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
SDL_bool SDL_WindowHasSurface(SDL_Window *window);
```

Function Parameters

window	the window to query

Return Value

Returns SDL_TRUE if there is a surface associated with the window, or SDL_FALSE otherwise.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetWindowSurface

SDL_WinRTGetDeviceFamily

Detects the device family of WinRT platform at runtime.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
SDL_WinRT_DeviceFamily SDL_WinRTGetDeviceFamily();
```

Return Value

Returns a value from the SDL_WinRT_DeviceFamily enum.

Version

This function is available since SDL 3.0.0.

SDL_WinRTGetFSPath

Retrieve a WinRT defined path on the local file system.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
const char * SDL_WinRTGetFSPath(SDL_WinRT_Path pathType);
```

Function Parameters

pathType	the type of path to retrieve, one of SDL_WinRT_Path

Return Value

Returns a UTF-8 string (8-bit, multi-byte) containing the path, or NULL if the path is not available for any reason; call SDL_GetError() for more information.

Remarks

Not all paths are available on all versions of Windows. This is especially true on Windows Phone. Check the documentation for the given SDL_WinRT_Path for more information on which path types are supported where.

Documentation on most app-specific path types on WinRT can be found on MSDN, at the URL:

https://msdn.microsoft.com/en-us/library/windows/apps/hh464917.aspx

Version

This function is available since SDL 3.0.0.

SDL_WriteIO

Write to an SDL_IOStream data stream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
size_t SDL_WriteIO(SDL_IOStream *context, const void *ptr, size_t size);
```

Function Parameters

context	a pointer to an SDL_IOStream structure
ptr	a pointer to a buffer containing data to write
size	the number of bytes to write

Return Value

Returns the number of bytes written, which will be less than `size` on error; call SDL_GetError() for more information.

Remarks

This function writes exactly `size` bytes from the area pointed at by `ptr` to the stream. If this fails for any reason, it'll return less than `size` to demonstrate how far the write progressed. On success, it returns `size`.

On error, this function still attempts to write as much as possible, so it might return a positive value less than the requested write size.

The caller can use SDL_GetIOStatus() to determine if the problem is recoverable, such as a non-blocking write that can simply be retried later, or a fatal error.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_IOprintf
- SDL_ReadIO
- SDL_SeekIO
- SDL_GetIOStatus

SDL_WriteS16BE

Use this function to write 16 bits in native format to an SDL_IOStream as big-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteS16BE(SDL_IOStream *dst, Sint16 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in big-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteS16LE

Use this function to write 16 bits in native format to an SDL_IOStream as little-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteS16LE(SDL_IOStream *dst, Sint16 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in little-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteS32BE

Use this function to write 32 bits in native format to an SDL_IOStream as big-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteS32BE(SDL_IOStream *dst, Sint32 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in big-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteS32LE

Use this function to write 32 bits in native format to an SDL_IOStream as little-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteS32LE(SDL_IOStream *dst, Sint32 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in little-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteS64BE

Use this function to write 64 bits in native format to an SDL_IOStream as big-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteS64BE(SDL_IOStream *dst, Sint64 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in big-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteS64LE

Use this function to write 64 bits in native format to an SDL_IOStream as little-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteS64LE(SDL_IOStream *dst, Sint64 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in little-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteStorageFile

Synchronously write a file from client memory into a storage container.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
int SDL_WriteStorageFile(SDL_Storage *storage, const char *path, const
    void *source, Uint64 length);
```

Function Parameters

storage	a storage container to write to
path	the relative path of the file to write
source	a client-provided buffer to write from
length	the length of the source buffer

Return Value

Returns 0 if the file was written, a negative value otherwise; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetStorageSpaceRemaining
- SDL_ReadStorageFile
- SDL_StorageReady

SDL_WriteU16BE

Use this function to write 16 bits in native format to an SDL_IOStream as big-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteU16BE(SDL_IOStream *dst, Uint16 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in big-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteU16LE

Use this function to write 16 bits in native format to an SDL_IOStream as little-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteU16LE(SDL_IOStream *dst, Uint16 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in little-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteU32BE

Use this function to write 32 bits in native format to an SDL_IOStream as big-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteU32BE(SDL_IOStream *dst, Uint32 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in big-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteU32LE

Use this function to write 32 bits in native format to an SDL_IOStream as little-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteU32LE(SDL_IOStream *dst, Uint32 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in little-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteU64BE

Use this function to write 64 bits in native format to an SDL_IOStream as big-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteU64BE(SDL_IOStream *dst, Uint64 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in big-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteU64LE

Use this function to write 64 bits in native format to an SDL_IOStream as little-endian data.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteU64LE(SDL_IOStream *dst, Uint64 value);
```

Function Parameters

dst	the stream to which data will be written
value	the data to be written, in native format

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Remarks

SDL byteswaps the data only if necessary, so the application always specifies native format, and the data written will be in little-endian format.

Version

This function is available since SDL 3.0.0.

SDL_WriteU8

Use this function to write a byte to an SDL_IOStream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
SDL_bool SDL_WriteU8(SDL_IOStream *dst, Uint8 value);
```

Function Parameters

dst	the SDL_IOStream to write to
value	the byte value to write

Return Value

Returns SDL_TRUE on successful write, SDL_FALSE on failure; call SDL_GetError() for more information.

Version

This function is available since SDL 3.0.0.

Chapter 2

SDL Macros

SDL_ANDROID_EXTERNAL_STORAGE_READ

See the official Android developer guide for more information: http://developer.android.com/guide/topics storage.html

Header File

Defined in SDL3/SDL_system.h

Syntax

```
#define SDL_ANDROID_EXTERNAL_STORAGE_READ 0x01
```

Version

This macro is available since SDL 3.0.0.

SDL_arraysize

The number of elements in an array.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_arraysize(array) (sizeof(array)/sizeof(array[0]))
```

Remarks

NOTE: This macro double-evaluates the argument, so you should never have side effects in the parameter.

Version

This macro is available since SDL 3.0.0.

SDL_assert

An assertion test that is normally performed only in debug builds.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
#define SDL_assert(condition) if (assertion_enabled && (condition)) {
    trigger_assertion; }
```

Remarks

This macro is enabled when the SDL_ASSERT_LEVEL is ¿= 2, otherwise it is disabled. This is meant to only do these tests in debug builds, so they can tend to be more expensive, and they are meant to bring everything to a halt when they fail, with the programmer there to assess the problem.

In short: you can sprinkle these around liberally and assume they will evaporate out of the build when building for end-users.

When assertions are disabled, this wraps `condition` in a `sizeof` operator, which means any function calls and side effects will not run, but the compiler will not complain about any otherwise-unused variables that are only referenced in the assertion.

One can set the environment variable "SDL_ASSERT" to one of several strings ("abort", "break", "retry", "ignore", "always_ignore") to force a default behavior, which may be desirable for automation purposes. If your platform requires GUI interfaces to happen on the main thread but you're debugging an assertion in a background thread, it might be desirable to set this to "break" so that your debugger takes control as soon as assert is triggered, instead of risking a bad UI interaction (deadlock, etc) in the application.

Note that SDL_ASSERT is an _environment variable_ and not an SDL hint! Please refer to your platform's documentation for how to set it!

Version

This macro is available since SDL 3.0.0.

SDL_assert_always

An assertion test that always performed.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
#define SDL_assert_always(condition) SDL_enabled_assert(condition)
```

Remarks

This macro is always enabled no matter what SDL_ASSERT_LEVEL is set to. You almost never want to use this, as it could trigger on an end-user's system, crashing your program.

One can set the environment variable "SDL_ASSERT" to one of several strings ("abort", "break", "retry", "ignore", "always_ignore") to force a default behavior, which may be desirable for automation purposes. If your platform requires GUI interfaces to happen on the main thread but you're debugging an assertion in a background thread, it might be desirable to set this to "break" so that your debugger takes control as soon as assert is triggered, instead of risking a bad UI interaction (deadlock, etc) in the application.

Note that SDL_ASSERT is an _environment variable_ and not an SDL hint! Please refer to your platform's documentation for how to set it!

Version

This macro is available since SDL 3.0.0.

SDL_ASSERT_LEVEL

The level of assertion aggressiveness.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
#define SDL_ASSERT_LEVEL SomeNumberBasedOnVariousFactors
```

Remarks

This value changes depending on compiler options and other preprocessor defines.

It is currently one of the following values, but future SDL releases might add more:

- 0: All SDL assertion macros are disabled.
- 1: Release settings: SDL_assert disabled, SDL_assert_release enabled.
- 2: Debug settings: SDL_assert and SDL_assert_release enabled.
- 3: Paranoid settings: All SDL assertion macros enabled, including SDL_assert_paranoid.

Version

This macro is available since SDL 3.0.0.

SDL_assert_paranoid

An assertion test that is performed only when built with paranoid settings.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
#define SDL_assert_paranoid(condition) SDL_disabled_assert(condition)
```

Remarks

This macro is enabled when the SDL_ASSERT_LEVEL is ¿= 3, otherwise it is disabled. This is a higher level than both release and debug, so these tests are meant to be expensive and only run when specifically looking for extremely unexpected failure cases in a special build.

When assertions are disabled, this wraps `condition` in a `sizeof` operator, which means any function calls and side effects will not run, but the compiler will not complain about any otherwise-unused variables that are only referenced in the assertion.

One can set the environment variable "SDL_ASSERT" to one of several strings ("abort", "break", "retry", "ignore", "always_ignore") to force a default behavior, which may be desirable for automation purposes. If your platform requires GUI interfaces to happen on the main thread but you're debugging an assertion in a background thread, it might be desirable to set this to "break" so that your debugger takes control as soon as assert is triggered, instead of risking a bad UI interaction (deadlock, etc) in the application.

Note that SDL_ASSERT is an _environment variable_ and not an SDL hint! Please refer to your platform's documentation for how to set it!

Version

This macro is available since SDL 3.0.0.

SDL_assert_release

An assertion test that is performed even in release builds.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
#define SDL_assert_release(condition) SDL_disabled_assert(condition)
```

Remarks

This macro is enabled when the SDL_ASSERT_LEVEL is ¿= 1, otherwise it is disabled. This is meant to be for tests that are cheap to make and extremely unlikely to fail; generally it is frowned upon to have an assertion failure in a release build, so these assertions generally need to be of more than life-and-death importance if there's a chance they might trigger. You should almost always consider handling these cases more gracefully than an assert allows.

When assertions are disabled, this wraps `condition` in a `sizeof` operator, which means any function calls and side effects will not run, but the compiler will not complain about any otherwise-unused variables that are only referenced in the assertion.

One can set the environment variable "SDL_ASSERT" to one of several strings ("abort", "break", "retry", "ignore", "always_ignore") to force a default behavior, which may be desirable for automation purposes. If your platform requires GUI interfaces to happen on the main thread but you're debugging an assertion in a background thread, it might be desirable to set this to "break" so that your debugger takes control as soon as assert is triggered, instead of risking a bad UI interaction (deadlock, etc) in the application.

Note that SDL_ASSERT is an _environment variable_ and not an SDL hint! Please refer to your platform's documentation for how to set it!

Version

This macro is available since SDL 3.0.0.

SDL_AtomicDecRef

Decrement an atomic variable used as a reference count.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
#define SDL_AtomicDecRef(a) (SDL_AtomicAdd(a, -1) == 1)
```

Return Value

Returns SDL_TRUE if the variable reached zero after decrementing, SDL_FALSE otherwise

Remarks

Note: If you don't know what this macro is for, you shouldn't use it!

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_AtomicIncRef

SDL_AtomicIncRef

Increment an atomic variable used as a reference count.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
#define SDL_AtomicIncRef(a) SDL_AtomicAdd(a, 1)
```

Return Value

Returns the previous value of the atomic variable.

Remarks

Note: If you don't know what this macro is for, you shouldn't use it!

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_AtomicDecRef

SDL_AUDIO_BITSIZE

Retrieve the size, in bits, from an SDL_AudioFormat.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_BITSIZE(x)     ((x) & SDL_AUDIO_MASK_BITSIZE)
```

Return Value

Returns data size in bits

Remarks

For example, `SDL_AUDIO_BITSIZE(SDL_AUDIO_S16)` returns 16.

Thread Safety

It is safe to call this macro from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_BYTESIZE

Retrieve the size, in bytes, from an SDL_AudioFormat.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_BYTESIZE(x)     (SDL_AUDIO_BITSIZE(x) / 8)
```

Return Value

Returns data size in bytes

Remarks

For example, `SDL_AUDIO_BYTESIZE(SDL_AUDIO_S16)` returns 2.

Thread Safety

It is safe to call this macro from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_DEVICE_DEFAULT_CAPTURE

A value used to request a default capture audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_DEVICE_DEFAULT_CAPTURE ((SDL_AudioDeviceID) 0xFFFFFFFE)
```

Remarks

Several functions that require an SDL_AudioDeviceID will accept this value to signify the app just wants the system to choose a default device instead of the app providing a specific one.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_DEVICE_DEFAULT_OUTPUT

A value used to request a default output audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_DEVICE_DEFAULT_OUTPUT ((SDL_AudioDeviceID) 0xFFFFFFFF)
```

Remarks

Several functions that require an SDL_AudioDeviceID will accept this value to signify the app just wants the system to choose a default device instead of the app providing a specific one.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_FRAMESIZE

Calculate the size of each audio frame (in bytes) from an SDL_AudioSpec.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_FRAMESIZE(x) (SDL_AUDIO_BYTESIZE((x).format) *
    (x).channels)
```

Return Value

Returns the number of bytes used per sample frame.

Remarks

This reports on the size of an audio sample frame: stereo Sint16 data (2 channels of 2 bytes each) would be 4 bytes per frame, for example.

Thread Safety

It is safe to call this macro from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_ISBIGENDIAN

Determine if an SDL_AudioFormat represents bigendian data.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_ISBIGENDIAN(x) ((x) & SDL_AUDIO_MASK_BIG_ENDIAN)
```

Return Value

Returns non-zero if format is bigendian, zero otherwise.

Remarks

For example, `SDL_AUDIO_ISBIGENDIAN(SDL_AUDIO_S16LE)` returns 0.

Thread Safety

It is safe to call this macro from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_ISFLOAT

Determine if an SDL_AudioFormat represents floating point data.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_ISFLOAT(x)      ((x) & SDL_AUDIO_MASK_FLOAT)
```

Return Value

Returns non-zero if format is floating point, zero otherwise.

Remarks

For example, `SDL_AUDIO_ISFLOAT(SDL_AUDIO_S16)` returns 0.

Thread Safety

It is safe to call this macro from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_ISINT

Determine if an SDL_AudioFormat represents integer data.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_ISINT(x)       (!SDL_AUDIO_ISFLOAT(x))
```

Return Value

Returns non-zero if format is integer, zero otherwise.

Remarks

For example, `SDL_AUDIO_ISINT(SDL_AUDIO_F32)` returns 0.

Thread Safety

It is safe to call this macro from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_ISLITTLEENDIAN

Determine if an SDL_AudioFormat represents littleendian data.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_ISLITTLEENDIAN(x) (!SDL_AUDIO_ISBIGENDIAN(x))
```

Return Value

Returns non-zero if format is littleendian, zero otherwise.

Remarks

For example, `SDL_AUDIO_ISLITTLEENDIAN(SDL_AUDIO_S16BE)` returns 0.

Thread Safety

It is safe to call this macro from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_ISSIGNED

Determine if an SDL_AudioFormat represents signed data.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_ISSIGNED(x)     ((x) & SDL_AUDIO_MASK_SIGNED)
```

Return Value

Returns non-zero if format is signed, zero otherwise.

Remarks

For example, `SDL_AUDIO_ISSIGNED(SDL_AUDIO_U8)` returns 0.

Thread Safety

It is safe to call this macro from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_AUDIO_ISUNSIGNED

Determine if an SDL_AudioFormat represents unsigned data.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_AUDIO_ISUNSIGNED(x)  (!SDL_AUDIO_ISSIGNED(x))
```

Return Value

Returns non-zero if format is unsigned, zero otherwise.

Remarks

For example, `SDL_AUDIO_ISUNSIGNED(SDL_AUDIO_S16)` returns 0.

Thread Safety

It is safe to call this macro from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_BUTTON

Used as a mask when testing buttons in buttonstate.

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
#define SDL_BUTTON(X)     (1 << ((X)-1))
```

Remarks

- Button 1: Left mouse button
- Button 2: Middle mouse button
- Button 3: Right mouse button
- Button 4: Side mouse button 1
- Button 5: Side mouse button 2

Version

This macro is available since SDL 3.0.0.

SDL_CACHELINE_SIZE

A guess for the cacheline size used for padding.

Header File

Defined in SDL3/SDL_cpuinfo.h

Syntax

```
#define SDL_CACHELINE_SIZE 128
```

Remarks

Most x86 processors have a 64 byte cache line. The 64-bit PowerPC processors have a 128 byte cache line. We use the larger value to be generally safe.

Version

This macro is available since SDL 3.0.0.

SDL_COMPILEDVERSION

This is the version number macro for the current SDL version.

Header File

Defined in SDL3/SDL_version.h

Syntax

```
#define SDL_COMPILEDVERSION \
    SDL_VERSIONNUM(SDL_MAJOR_VERSION, SDL_MINOR_VERSION, SDL_PATCHLEVEL)
```

Version

This macro is available since SDL 3.0.0.

SDL_CompilerBarrier

Mark a compiler barrier.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
#define SDL_CompilerBarrier() DoCompilerSpecificReadWriteBarrier()
```

Remarks

A compiler barrier prevents the compiler from reordering reads and writes to globally visible variables across the call.

This macro only prevents the compiler from reordering reads and writes, it does not prevent the CPU from reordering reads and writes. However, all of the atomic operations that modify memory are full memory barriers.

Thread Safety

Obviously this macro is safe to use from any thread at any time, but if you find yourself needing this, you are probably dealing with some very sensitive code; be careful!

Version

This macro is available since SDL 3.0.0.

SDL_CPUPauseInstruction

A macro to insert a CPU-specific "pause" instruction into the program.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
#define SDL_CPUPauseInstruction()
    DoACPUPauseInACompilerAndArchitectureSpecificWay
```

Remarks

This can be useful in busy-wait loops, as it serves as a hint to the CPU as to the program's intent; some CPUs can use this to do more efficient processing. On some platforms, this doesn't do anything, so using this macro might just be a harmless no-op.

Note that if you are busy-waiting, there are often more-efficient approaches with other synchronization primitives: mutexes, semaphores, condition variables, etc.

Thread Safety

This macro is safe to use from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_FALSE

A boolean false.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_FALSE 0
```

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_bool

SDL_HAPTIC_AUTOCENTER

Device can set autocenter.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_AUTOCENTER (1u<<17)
```

Remarks

Device supports setting autocenter.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_SetHapticAutocenter

SDL_HAPTIC_CARTESIAN

Uses cartesian coordinates for the direction.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_CARTESIAN 1
```

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticDirection

SDL_HAPTIC_CONSTANT

Constant effect supported.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_CONSTANT  (1u<<0)
```

Remarks

Constant haptic effect.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticCondition

SDL_HAPTIC_CUSTOM

Custom effect is supported.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_CUSTOM    (1u<<15)
```

Remarks

User defined custom haptic effect.

Version

This macro is available since SDL 3.0.0.

SDL_HAPTIC_DAMPER

Damper effect supported - uses axes velocity.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_DAMPER    (1u<<8)
```

Remarks

Condition haptic effect that simulates dampening. Effect is based on the axes velocity.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticCondition

SDL_HAPTIC_FRICTION

Friction effect supported - uses axes movement.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_FRICTION  (1u<<10)
```

Remarks

Condition haptic effect that simulates friction. Effect is based on the axes movement.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticCondition

SDL_HAPTIC_GAIN

Device can set global gain.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_GAIN    (1u<<16)
```

Remarks

Device supports setting the global gain.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_SetHapticGain

SDL_HAPTIC_INERTIA

Inertia effect supported - uses axes acceleration.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_INERTIA   (1u<<9)
```

Remarks

Condition haptic effect that simulates inertia. Effect is based on the axes acceleration.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticCondition

SDL_HAPTIC_INFINITY

Used to play a device an infinite number of times.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_INFINITY 4294967295U
```

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_RunHapticEffect

SDL_HAPTIC_LEFTRIGHT

Left/Right effect supported.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_LEFTRIGHT (1u<<11)
```

Remarks

Haptic effect for direct control over high/low frequency motors.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticLeftRight

SDL_HAPTIC_PAUSE

Device can be paused.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_PAUSE   (1u<<19)
```

Remarks

Devices supports being paused.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_PauseHaptic
- SDL_ResumeHaptic

SDL_HAPTIC_POLAR

Uses polar coordinates for the direction.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_POLAR   0
```

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticDirection

SDL_HAPTIC_RAMP

Ramp effect supported.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_RAMP      (1u<<6)
```

Remarks

Ramp haptic effect.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticRamp

SDL_HAPTIC_RESERVED1

Reserved for future use

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_RESERVED1 (1u<<12)
```

Version

This macro is available since SDL 3.0.0.

SDL_HAPTIC_SAWTOOTHDOWN

Sawtoothdown wave effect supported.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_SAWTOOTHDOWN (1u<<5)
```

Remarks

Periodic haptic effect that simulates saw tooth down waves.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticPeriodic

SDL_HAPTIC_SAWTOOTHUP

Sawtoothup wave effect supported.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_SAWTOOTHUP (1u<<4)
```

Remarks

Periodic haptic effect that simulates saw tooth up waves.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticPeriodic

SDL_HAPTIC_SINE

Sine wave effect supported.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_SINE      (1u<<1)
```

Remarks

Periodic haptic effect that simulates sine waves.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticPeriodic

SDL_HAPTIC_SPHERICAL

Uses spherical coordinates for the direction.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_SPHERICAL 2
```

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticDirection

SDL_HAPTIC_SPRING

Spring effect supported - uses axes position.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_SPRING    (1u<<7)
```

Remarks

Condition haptic effect that simulates a spring. Effect is based on the axes position.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticCondition

SDL_HAPTIC_SQUARE

Square wave effect supported.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_SQUARE    (1<<2)
```

Remarks

Periodic haptic effect that simulates square waves.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticPeriodic

SDL_HAPTIC_STATUS

Device can be queried for effect status.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_STATUS  (1u<<18)
```

Remarks

Device supports querying effect status.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_GetHapticEffectStatus

SDL_HAPTIC_STEERING_AXIS

Use this value to play an effect on the steering wheel axis.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_STEERING_AXIS 3
```

Remarks

This provides better compatibility across platforms and devices as SDL will guess the correct axis.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticDirection

SDL_HAPTIC_TRIANGLE

Triangle wave effect supported.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
#define SDL_HAPTIC_TRIANGLE  (1u<<3)
```

Remarks

Periodic haptic effect that simulates triangular waves.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_HapticPeriodic

SDL_HINT_ALLOW_ALT_TAB_WHILE_GRABBED

Specify the behavior of Alt+Tab while the keyboard is grabbed.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_ALLOW_ALT_TAB_WHILE_GRABBED
    "SDL_ALLOW_ALT_TAB_WHILE_GRABBED"
```

Remarks

By default, SDL emulates Alt+Tab functionality while the keyboard is grabbed and your window is full-screen. This prevents the user from getting stuck in your application if you've enabled keyboard grab.

The variable can be set to the following values:

- "0": SDL will not handle Alt+Tab. Your application is responsible for handling Alt+Tab while the keyboard is grabbed.
- "1": SDL will minimize your window when Alt+Tab is pressed (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_ANDROID_ALLOW_RECREATE_ACTIVITY

A variable to control whether the SDL activity is allowed to be re-created.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_ANDROID_ALLOW_RECREATE_ACTIVITY
    "SDL_ANDROID_ALLOW_RECREATE_ACTIVITY"
```

Remarks

If this hint is true, the activity can be recreated on demand by the OS, and Java static data and C++ static data remain with their current values. If this hint is false, then SDL will call exit() when you return from your main function and the application will be terminated and then started fresh each time.

The variable can be set to the following values:

- "0": The application starts fresh at each launch. (default)
- "1": The application activity can be recreated by the OS.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_ANDROID_BLOCK_ON_PAUSE

A variable to control whether the event loop will block itself when the app is paused.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_ANDROID_BLOCK_ON_PAUSE "SDL_ANDROID_BLOCK_ON_PAUSE"
```

Remarks

The variable can be set to the following values:

- "0": Non blocking.
- "1": Blocking. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_ANDROID_BLOCK_ON_PAUSE_PAUSEAUDIO

A variable to control whether SDL will pause audio in background.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_ANDROID_BLOCK_ON_PAUSE_PAUSEAUDIO
    "SDL_ANDROID_BLOCK_ON_PAUSE_PAUSEAUDIO"
```

Remarks

The variable can be set to the following values:

- "0": Not paused, requires that SDL_HINT_ANDROID_BLOCK_ON_PAUSE be set to "0"
- "1": Paused. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_ANDROID_TRAP_BACK_BUTTON

A variable to control whether we trap the Android back button to handle it manually.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_ANDROID_TRAP_BACK_BUTTON "SDL_ANDROID_TRAP_BACK_BUTTON"
```

Remarks

This is necessary for the right mouse button to work on some Android devices, or to be able to trap the back button for use in your code reliably. If this hint is true, the back button will show up as an SDL_EVENT_KEY_DOWN / SDL_EVENT_KEY_UP pair with a keycode of SDL_SCANCODE_AC_BACK.

The variable can be set to the following values:

- "0": Back button will be handled as usual for system. (default)
- "1": Back button will be trapped, allowing you to handle the key press manually. (This will also let right mouse click work on systems where the right mouse button functions as back.)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_APP_ID

A variable setting the app ID string.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_APP_ID   "SDL_APP_ID"
```

Remarks

This string is used by desktop compositors to identify and group windows together, as well as match applications with associated desktop settings and icons.

On Wayland this corresponds to the "app ID" window property and on X11 this corresponds to the WM_CLASS property. Windows inherit the value of this hint at creation time. Changing this hint after a window has been created will not change the app ID or class of existing windows.

For *nix platforms, this string should be formatted in reverse-DNS notation and follow some basic rules to be valid:

- The application ID must be composed of two or more elements separated by a period (.) character.
- Each element must contain one or more of the alphanumeric characters (A-Z, a-z, 0-9) plus underscore (_) and hyphen (-) and must not start with a digit. Note that hyphens, while technically allowed, should not be used if possible, as they are not supported by all components that use the ID, such as D-Bus. For maximum compatibility, replace hyphens with an underscore.
- The empty string is not a valid element (ie: your application ID may not start or end with a period and it is not valid to have two periods in a row).
- The entire ID must be less than 255 characters in length.

Examples of valid app ID strings:

- org.MyOrg.MyApp
- com.your_company.your_app

Desktops such as GNOME and KDE require that the app ID string matches your application's .desktop file name (e.g. if the app ID string is 'org.MyOrg.MyApp', your application's .desktop file should be named 'org.MyOrg.MyApp.desktop').

If you plan to package your application in a container such as Flatpak, the app ID should match the name of your Flatpak container as well.

If not set, SDL will attempt to use the application executable name. If the executable name cannot be retrieved, the generic string "SDL_App" will be used.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_APP_NAME

Specify an application name.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_APP_NAME "SDL_APP_NAME"
```

Remarks

This hint lets you specify the application name sent to the OS when required. For example, this will often appear in volume control applets for audio streams, and in lists of applications which are inhibiting the screensaver. You should use a string that describes your program ("My Game 2: The Revenge")

Setting this to "" or leaving it unset will have SDL use a reasonable default: probably the application's name or "SDL Application" if SDL doesn't have any better information.

Note that, for audio streams, this can be overridden with SDL_HINT_AUDIO_DEVICE_APP_NAME.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_APPLE_TV_CONTROLLER_UI_EVENTS

A variable controlling whether controllers used with the Apple TV generate UI events.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_APPLE_TV_CONTROLLER_UI_EVENTS
    "SDL_APPLE_TV_CONTROLLER_UI_EVENTS"
```

Remarks

When UI events are generated by controller input, the app will be backgrounded when the Apple TV remote's menu button is pressed, and when the pause or B buttons on gamepads are pressed.

More information about properly making use of controllers for the Apple TV can be found here: https://developer.apple.com/tvos/human-interface-guidelines/remote-and-controllers/

The variable can be set to the following values:

- "0": Controller input does not generate UI events. (default)
- "1": Controller input generates UI events.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_APPLE_TV_REMOTE_ALLOW_ROTATION

A variable controlling whether the Apple TV remote's joystick axes will automatically match the rotation of the remote.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_APPLE_TV_REMOTE_ALLOW_ROTATION
    "SDL_APPLE_TV_REMOTE_ALLOW_ROTATION"
```

Remarks

The variable can be set to the following values:

- "0": Remote orientation does not affect joystick axes. (default)
- "1": Joystick axes are based on the orientation of the remote.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_AUDIO_CATEGORY

A variable controlling the audio category on iOS and macOS.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_AUDIO_CATEGORY "SDL_AUDIO_CATEGORY"
```

Remarks

The variable can be set to the following values:

- "ambient": Use the AVAudioSessionCategoryAmbient audio category, will be muted by the phone mute switch (default)
- "playback": Use the AVAudioSessionCategoryPlayback category.

For more information, see Apple's documentation: https://developer.apple.com/library/content/documentation/A
This hint should be set before an audio device is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_AUDIO_DEVICE_APP_NAME

Specify an application name for an audio device.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_AUDIO_DEVICE_APP_NAME "SDL_AUDIO_DEVICE_APP_NAME"
```

Remarks

Some audio backends (such as PulseAudio) allow you to describe your audio stream. Among other things, this description might show up in a system control panel that lets the user adjust the volume on specific audio streams instead of using one giant master volume slider.

This hints lets you transmit that information to the OS. The contents of this hint are used while opening an audio device. You should use a string that describes your program ("My Game 2: The Revenge")

Setting this to "" or leaving it unset will have SDL use a reasonable default: this will be the name set with SDL_HINT_APP_NAME, if that hint is set. Otherwise, it'll probably the application's name or "SDL Application" if SDL doesn't have any better information.

This hint should be set before an audio device is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_AUDIO_DEVICE_SAMPLE_FRAMES

A variable controlling device buffer size.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_AUDIO_DEVICE_SAMPLE_FRAMES
    "SDL_AUDIO_DEVICE_SAMPLE_FRAMES"
```

Remarks

This hint is an integer ¿ 0, that represents the size of the device's buffer in sample frames (stereo audio data in 16-bit format is 4 bytes per sample frame, for example).

SDL3 generally decides this value on behalf of the app, but if for some reason the app needs to dictate this (because they want either lower latency or higher throughput AND ARE WILLING TO DEAL WITH what that might require of the app), they can specify it.

SDL will try to accomodate this value, but there is no promise you'll get the buffer size requested. Many platforms won't honor this request at all, or might adjust it.

This hint should be set before an audio device is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_AUDIO_DEVICE_STREAM_NAME

Specify an audio stream name for an audio device.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_AUDIO_DEVICE_STREAM_NAME "SDL_AUDIO_DEVICE_STREAM_NAME"
```

Remarks

Some audio backends (such as PulseAudio) allow you to describe your audio stream. Among other things, this description might show up in a system control panel that lets the user adjust the volume on specific audio streams instead of using one giant master volume slider.

This hints lets you transmit that information to the OS. The contents of this hint are used while opening an audio device. You should use a string that describes your what your program is playing ("audio stream" is probably sufficient in many cases, but this could be useful for something like "team chat" if you have a headset playing VoIP audio separately).

Setting this to "" or leaving it unset will have SDL use a reasonable default: "audio stream" or something similar.

Note that while this talks about audio streams, this is an OS-level concept, so it applies to a physical audio device in this case, and not an SDL_AudioStream, nor an SDL logical audio device.

This hint should be set before an audio device is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_AUDIO_DEVICE_STREAM_ROLE

Specify an application role for an audio device.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_AUDIO_DEVICE_STREAM_ROLE "SDL_AUDIO_DEVICE_STREAM_ROLE"
```

Remarks

Some audio backends (such as Pipewire) allow you to describe the role of your audio stream. Among other things, this description might show up in a system control panel or software for displaying and manipulating media playback/capture graphs.

This hints lets you transmit that information to the OS. The contents of this hint are used while opening an audio device. You should use a string that describes your what your program is playing (Game, Music, Movie, etc...).

Setting this to "" or leaving it unset will have SDL use a reasonable default: "Game" or something similar.

Note that while this talks about audio streams, this is an OS-level concept, so it applies to a physical audio device in this case, and not an SDL_AudioStream, nor an SDL logical audio device.

This hint should be set before an audio device is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_AUDIO_DRIVER

A variable that specifies an audio backend to use.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_AUDIO_DRIVER "SDL_AUDIO_DRIVER"
```

Remarks

By default, SDL will try all available audio backends in a reasonable order until it finds one that can work, but this hint allows the app or user to force a specific driver, such as "pipewire" if, say, you are on PulseAudio but want to try talking to the lower level instead.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_AUDIO_INCLUDE_MONITORS

A variable that causes SDL to not ignore audio "monitors".

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_AUDIO_INCLUDE_MONITORS "SDL_AUDIO_INCLUDE_MONITORS"
```

Remarks

This is currently only used by the PulseAudio driver.

By default, SDL ignores audio devices that aren't associated with physical hardware. Changing this hint to "1" will expose anything SDL sees that appears to be an audio source or sink. This will add "devices" to the list that the user probably doesn't want or need, but it can be useful in scenarios where you want to hook up SDL to some sort of virtual device, etc.

The variable can be set to the following values:

- "0": Audio monitor devices will be ignored. (default)
- "1": Audio monitor devices will show up in the device list.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_AUTO_UPDATE_JOYSTICKS

A variable controlling whether SDL updates joystick state when getting input events.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_AUTO_UPDATE_JOYSTICKS "SDL_AUTO_UPDATE_JOYSTICKS"
```

Remarks

The variable can be set to the following values:

- "0": You'll call SDL_UpdateJoysticks() manually.
- "1": SDL will automatically call SDL_UpdateJoysticks(). (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_AUTO_UPDATE_SENSORS

A variable controlling whether SDL updates sensor state when getting input events.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_AUTO_UPDATE_SENSORS "SDL_AUTO_UPDATE_SENSORS"
```

Remarks

The variable can be set to the following values:

- "0": You'll call SDL_UpdateSensors() manually.
- "1": SDL will automatically call SDL_UpdateSensors(). (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_BMP_SAVE_LEGACY_FORMAT

Prevent SDL from using version 4 of the bitmap header when saving BMPs.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_BMP_SAVE_LEGACY_FORMAT "SDL_BMP_SAVE_LEGACY_FORMAT"
```

Remarks

The bitmap header version 4 is required for proper alpha channel support and SDL will use it when required. Should this not be desired, this hint can force the use of the 40 byte header version which is supported everywhere.

The variable can be set to the following values:

- "0": Surfaces with a colorkey or an alpha channel are saved to a 32-bit BMP file with an alpha mask. SDL will use the bitmap header version 4 and set the alpha mask accordingly. (default)
- "1": Surfaces with a colorkey or an alpha channel are saved to a 32-bit BMP file without an alpha mask. The alpha channel data will be in the file, but applications are going to ignore it.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_CAMERA_DRIVER

A variable that decides what camera backend to use.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_CAMERA_DRIVER "SDL_CAMERA_DRIVER"
```

Remarks

By default, SDL will try all available camera backends in a reasonable order until it finds one that can work, but this hint allows the app or user to force a specific target, such as "directshow" if, say, you are on Windows Media Foundations but want to try DirectShow instead.

The default value is unset, in which case SDL will try to figure out the best camera backend on your behalf. This hint needs to be set before SDL_Init() is called to be useful.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_CPU_FEATURE_MASK

A variable that limits what CPU features are available.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_CPU_FEATURE_MASK "SDL_CPU_FEATURE_MASK"
```

Remarks

By default, SDL marks all features the current CPU supports as available. This hint allows to limit these to a subset.

When the hint is unset, or empty, SDL will enable all detected CPU features.

The variable can be set to a comma separated list containing the following items:

- "all"
- "altivec"
- "sse"
- "sse2"
- "sse3"
- "sse41"
- "sse42"
- "avx"
- "avx2"
- "avx512f"
- "arm-simd"
- "neon"
- "lsx"
- "lasx"

The items can be prefixed by '+'/'-' to add/remove features.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_DISPLAY_USABLE_BOUNDS

Override for SDL_GetDisplayUsableBounds().

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_DISPLAY_USABLE_BOUNDS "SDL_DISPLAY_USABLE_BOUNDS"
```

Remarks

If set, this hint will override the expected results for SDL_GetDisplayUsableBounds() for display index 0. Generally you don't want to do this, but this allows an embedded system to request that some of the screen be reserved for other uses when paired with a well-behaved application.

The contents of this hint must be 4 comma-separated integers, the first is the bounds x, then y, width and height, in that order.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_EMSCRIPTEN_ASYNCIFY

Disable giving back control to the browser automatically when running with asyncify.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_EMSCRIPTEN_ASYNCIFY "SDL_EMSCRIPTEN_ASYNCIFY"
```

Remarks

With -s ASYNCIFY, SDL calls emscripten_sleep during operations such as refreshing the screen or polling events.

This hint only applies to the emscripten platform.

The variable can be set to the following values:

- "0": Disable emscripten_sleep calls (if you give back browser control manually or use asyncify for other purposes).
- "1": Enable emscripten_sleep calls. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_EMSCRIPTEN_CANVAS_SELECTOR

Specify the CSS selector used for the "default" window/canvas.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_EMSCRIPTEN_CANVAS_SELECTOR
    "SDL_EMSCRIPTEN_CANVAS_SELECTOR"
```

Remarks

This hint only applies to the emscripten platform.

The default value is "#canvas"

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_EMSCRIPTEN_KEYBOARD_ELEMENT

Override the binding element for keyboard inputs for Emscripten builds.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_EMSCRIPTEN_KEYBOARD_ELEMENT
    "SDL_EMSCRIPTEN_KEYBOARD_ELEMENT"
```

Remarks

This hint only applies to the emscripten platform.

The variable can be one of:

- "#window": the javascript window object (default)
- "#document": the javascript document object
- "#screen": the javascript window.screen object
- "#canvas": the WebGL canvas element
- any other string without a leading # sign applies to the element on the page with that ID.

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_ENABLE_SCREEN_KEYBOARD

A variable that controls whether the on-screen keyboard should be shown when text input is active.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_ENABLE_SCREEN_KEYBOARD "SDL_ENABLE_SCREEN_KEYBOARD"
```

Remarks

The variable can be set to the following values:

- "auto": The on-screen keyboard will be shown if there is no physical keyboard attached. (default)
- "0": Do not show the on-screen keyboard.
- "1": Show the on-screen keyboard, if available.

This hint must be set before SDL_StartTextInput() is called

Version

This hint is available since SDL 3.0.0.

SDL_HINT_EVENT_LOGGING

A variable controlling verbosity of the logging of SDL events pushed onto the internal queue.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_EVENT_LOGGING "SDL_EVENT_LOGGING"
```

Remarks

The variable can be set to the following values, from least to most verbose:

- "0": Don't log any events. (default)
- "1": Log most events (other than the really spammy ones).
- "2": Include mouse and finger motion events.

This is generally meant to be used to debug SDL itself, but can be useful for application developers that need better visibility into what is going on in the event queue. Logged events are sent through SDL_Log(), which means by default they appear on stdout on most platforms or maybe OutputDebugString() on Windows, and can be funneled by the app with SDL_SetLogOutputFunction(), etc.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_FILE_DIALOG_DRIVER

A variable that specifies a dialog backend to use.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_FILE_DIALOG_DRIVER "SDL_FILE_DIALOG_DRIVER"
```

Remarks

By default, SDL will try all available dialog backends in a reasonable order until it finds one that can work, but this hint allows the app or user to force a specific target.

If the specified target does not exist or is not available, the dialog-related function calls will fail.

This hint currently only applies to platforms using the generic "Unix" dialog implementation, but may be extended to more platforms in the future. Note that some Unix and Unix-like platforms have their own implementation, such as macOS and Haiku.

The variable can be set to the following values:

- NULL: Select automatically (default, all platforms)
- "portal": Use XDG Portals through DBus (Unix only)
- "zenity": Use the Zenity program (Unix only)

More options may be added in the future.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_FORCE_RAISEWINDOW

A variable controlling whether raising the window should be done more forcefully.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_FORCE_RAISEWINDOW "SDL_FORCE_RAISEWINDOW"
```

Remarks

The variable can be set to the following values:

- "0": Honor the OS policy for raising windows. (default)
- "1": Force the window to be raised, overriding any OS policy.

At present, this is only an issue under MS Windows, which makes it nearly impossible to programmatically move a window to the foreground, for "security" reasons. See http://stackoverflow.com/a/34414846 for a discussion.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_FRAMEBUFFER_ACCELERATION

A variable controlling how 3D acceleration is used to accelerate the SDL screen surface.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_FRAMEBUFFER_ACCELERATION "SDL_FRAMEBUFFER_ACCELERATION"
```

Remarks

SDL can try to accelerate the SDL screen surface by using streaming textures with a 3D rendering engine. This variable controls whether and how this is done.

The variable can be set to the following values:

- "0": Disable 3D acceleration
- "1": Enable 3D acceleration, using the default renderer. (default)
- "X": Enable 3D acceleration, using X where X is one of the valid rendering drivers. (e.g. "direct3d", "opengl", etc.)

This hint should be set before calling SDL_GetWindowSurface()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GAMECONTROLLER_IGNORE_DEVICES

A variable containing a list of devices to skip when scanning for game controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GAMECONTROLLER_IGNORE_DEVICES
    "SDL_GAMECONTROLLER_IGNORE_DEVICES"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g.

0xAAAA/0xBBBB,0xCCCC/0xDDDD

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GAMECONTROLLER_IGNORE_DEVICES_EXCE

If set, all devices will be skipped when scanning for game controllers except for the ones listed in this variable.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GAMECONTROLLER_IGNORE_DEVICES_EXCEPT
    "SDL_GAMECONTROLLER_IGNORE_DEVICES_EXCEPT"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g.

0xAAAA/0xBBBB,0xCCCC/0xDDDD

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GAMECONTROLLER_SENSOR_FUSION

A variable that controls whether the device's built-in accelerometer and gyro should be used as sensors for gamepads.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GAMECONTROLLER_SENSOR_FUSION
    "SDL_GAMECONTROLLER_SENSOR_FUSION"
```

Remarks

The variable can be set to the following values:

- "0": Sensor fusion is disabled
- "1": Sensor fusion is enabled for all controllers that lack sensors

Or the variable can be a comma separated list of USB VID/PID pairs in hexadecimal form, e.g.

0xAAAA/0xBBBB,0xCCCC/0xDDDD

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint should be set before a gamepad is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GAMECONTROLLERCONFIG

A variable that lets you manually hint extra gamecontroller db entries.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GAMECONTROLLERCONFIG "SDL_GAMECONTROLLERCONFIG"
```

Remarks

The variable should be newline delimited rows of gamecontroller config data, see SDL_gamepad.h

You can update mappings after SDL is initialized with SDL_GetGamepadMappingForGUID() and SDL_AddGamepadMapping()

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GAMECONTROLLERCONFIG_FILE

A variable that lets you provide a file with extra gamecontroller db entries.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GAMECONTROLLERCONFIG_FILE
    "SDL_GAMECONTROLLERCONFIG_FILE"
```

Remarks

The file should contain lines of gamecontroller config data, see SDL_gamepad.h

You can update mappings after SDL is initialized with SDL_GetGamepadMappingForGUID() and SDL_AddGamepadMapping()

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GAMECONTROLLERTYPE

A variable that overrides the automatic controller type detection.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GAMECONTROLLERTYPE "SDL_GAMECONTROLLERTYPE"
```

Remarks

The variable should be comma separated entries, in the form: VID/PID=type
The VID and PID should be hexadecimal with exactly 4 digits, e.g. 0x00fd
This hint affects what low level protocol is used with the HIDAPI driver.
The variable can be set to the following values:

- "Xbox360"
- "XboxOne"
- "PS3"
- "PS4"
- "PS5"
- "SwitchPro"

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GDK_TEXTINPUT_DEFAULT_TEXT

This variable sets the default text of the TextInput window on GDK platforms.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GDK_TEXTINPUT_DEFAULT_TEXT
    "SDL_GDK_TEXTINPUT_DEFAULT_TEXT"
```

Remarks

This hint is available only if SDL_GDK_TEXTINPUT defined.

This hint should be set before calling SDL_StartTextInput()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GDK_TEXTINPUT_DESCRIPTION

This variable sets the description of the TextInput window on GDK platforms.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GDK_TEXTINPUT_DESCRIPTION
    "SDL_GDK_TEXTINPUT_DESCRIPTION"
```

Remarks

This hint is available only if SDL_GDK_TEXTINPUT defined.

This hint should be set before calling SDL_StartTextInput()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GDK_TEXTINPUT_MAX_LENGTH

This variable sets the maximum input length of the TextInput window on GDK platforms.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GDK_TEXTINPUT_MAX_LENGTH "SDL_GDK_TEXTINPUT_MAX_LENGTH"
```

Remarks

The value must be a stringified integer, for example "10" to allow for up to 10 characters of text input.

This hint is available only if SDL_GDK_TEXTINPUT defined.

This hint should be set before calling SDL_StartTextInput()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GDK_TEXTINPUT_SCOPE

This variable sets the input scope of the TextInput window on GDK platforms.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GDK_TEXTINPUT_SCOPE "SDL_GDK_TEXTINPUT_SCOPE"
```

Remarks

Set this hint to change the XGameUiTextEntryInputScope value that will be passed to the window creation function. The value must be a stringified integer, for example "0" for XGameUiTextEntryInputScope::Default.

This hint is available only if SDL_GDK_TEXTINPUT defined.

This hint should be set before calling SDL_StartTextInput()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_GDK_TEXTINPUT_TITLE

This variable sets the title of the TextInput window on GDK platforms.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_GDK_TEXTINPUT_TITLE "SDL_GDK_TEXTINPUT_TITLE"
```

Remarks

This hint is available only if SDL_GDK_TEXTINPUT defined.

This hint should be set before calling SDL_StartTextInput()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_HIDAPI_ENUMERATE_ONLY_CONTROLLERS

A variable to control whether SDL_hid_enumerate() enumerates all HID devices or only controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_HIDAPI_ENUMERATE_ONLY_CONTROLLERS
    "SDL_HIDAPI_ENUMERATE_ONLY_CONTROLLERS"
```

Remarks

The variable can be set to the following values:

- "0": SDL_hid_enumerate() will enumerate all HID devices.
- "1": SDL_hid_enumerate() will only enumerate controllers. (default)

By default SDL will only enumerate controllers, to reduce risk of hanging or crashing on devices with bad drivers and avoiding macOS keyboard capture permission prompts.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_HIDAPI_IGNORE_DEVICES

A variable containing a list of devices to ignore in SDL_hid_enumerate().

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_HIDAPI_IGNORE_DEVICES "SDL_HIDAPI_IGNORE_DEVICES"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

For example, to ignore the Shanwan DS3 controller and any Valve controller, you might use the string "0x2563/0x0523,0x28de/0x0000"

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_IME_INTERNAL_EDITING

A variable to control whether certain IMEs should handle text editing internally instead of sending SDL_EVENT_TEXT_EDITING events.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_IME_INTERNAL_EDITING "SDL_IME_INTERNAL_EDITING"
```

Remarks

The variable can be set to the following values:

- "0": SDL_EVENT_TEXT_EDITING events are sent, and it is the application's responsibility to render the text from these events and differentiate it somehow from committed text. (default)
- "1": If supported by the IME then SDL_EVENT_TEXT_EDITING events are not sent, and text that is being composed will be rendered in its own UI.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_IME_SHOW_UI

A variable to control whether certain IMEs should show native UI components (such as the Candidate List) instead of suppressing them.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_IME_SHOW_UI "SDL_IME_SHOW_UI"
```

Remarks

The variable can be set to the following values:

- "0": Native UI components are not display. (default)
- "1": Native UI components are displayed.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_IOS_HIDE_HOME_INDICATOR

A variable controlling whether the home indicator bar on iPhone X should be hidden.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_IOS_HIDE_HOME_INDICATOR "SDL_IOS_HIDE_HOME_INDICATOR"
```

Remarks

The variable can be set to the following values:

- "0": The indicator bar is not hidden. (default for windowed applications)
- "1": The indicator bar is hidden and is shown when the screen is touched (useful for movie playback applications).
- "2": The indicator bar is dim and the first swipe makes it visible and the second swipe performs the "home" action. (default for fullscreen applications)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_ALLOW_BACKGROUND_EVENTS

A variable that lets you enable joystick (and gamecontroller) events even when your app is in the background.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_ALLOW_BACKGROUND_EVENTS
    "SDL_JOYSTICK_ALLOW_BACKGROUND_EVENTS"
```

Remarks

The variable can be set to the following values:

- "0": Disable joystick & gamecontroller input events when the application is in the background. (default)
- "1": Enable joystick & gamecontroller input events when the application is in the background.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_ARCADESTICK_DEVICES

A variable containing a list of arcade stick style controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_ARCADESTICK_DEVICES
    "SDL_JOYSTICK_ARCADESTICK_DEVICES"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_ARCADESTICK_DEVICES_EXCLUDED

A variable containing a list of devices that are not arcade stick style controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_ARCADESTICK_DEVICES_EXCLUDED
    "SDL_JOYSTICK_ARCADESTICK_DEVICES_EXCLUDED"
```

Remarks

This will override SDL_HINT_JOYSTICK_ARCADESTICK_DEVICES and the built in device list.

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_BLACKLIST_DEVICES

A variable containing a list of devices that should not be considerd joysticks.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_BLACKLIST_DEVICES
    "SDL_JOYSTICK_BLACKLIST_DEVICES"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_BLACKLIST_DEVICES_EXCLUDED

A variable containing a list of devices that should be considered joysticks.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_BLACKLIST_DEVICES_EXCLUDED
    "SDL_JOYSTICK_BLACKLIST_DEVICES_EXCLUDED"
```

Remarks

This will override SDL_HINT_JOYSTICK_BLACKLIST_DEVICES and the built in device list.

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_DEVICE

A variable containing a comma separated list of devices to open as joysticks.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_DEVICE "SDL_JOYSTICK_DEVICE"
```

Remarks

This variable is currently only used by the Linux joystick driver.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_DIRECTINPUT

A variable controlling whether DirectInput should be used for controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_DIRECTINPUT "SDL_JOYSTICK_DIRECTINPUT"
```

Remarks

The variable can be set to the following values:

- "0": Disable DirectInput detection.
- "1": Enable DirectInput detection. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_FLIGHTSTICK_DEVICES

A variable containing a list of flightstick style controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_FLIGHTSTICK_DEVICES
    "SDL_JOYSTICK_FLIGHTSTICK_DEVICES"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of @file, in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_FLIGHTSTICK_DEVICES_EXCLUDED

A variable containing a list of devices that are not flightstick style controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_FLIGHTSTICK_DEVICES_EXCLUDED
    "SDL_JOYSTICK_FLIGHTSTICK_DEVICES_EXCLUDED"
```

Remarks

This will override SDL_HINT_JOYSTICK_FLIGHTSTICK_DEVICES and the built in device list.

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_GAMECUBE_DEVICES

A variable containing a list of devices known to have a GameCube form factor.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_GAMECUBE_DEVICES
    "SDL_JOYSTICK_GAMECUBE_DEVICES"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_GAMECUBE_DEVICES_EXCLUDED

A variable containing a list of devices known not to have a GameCube form factor.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_GAMECUBE_DEVICES_EXCLUDED
    "SDL_JOYSTICK_GAMECUBE_DEVICES_EXCLUDED"
```

Remarks

This will override SDL_HINT_JOYSTICK_GAMECUBE_DEVICES and the built in device list.

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI

A variable controlling whether the HIDAPI joystick drivers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI "SDL_JOYSTICK_HIDAPI"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI drivers are not used.
- "1": HIDAPI drivers are used. (default)

This variable is the default for all drivers, but can be overridden by the hints for specific drivers below.

This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_COMBINE_JOY_CONS

A variable controlling whether Nintendo Switch Joy-Con controllers will be combined into a single Pro-like controller when using the HIDAPI driver.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_COMBINE_JOY_CONS
    "SDL_JOYSTICK_HIDAPI_COMBINE_JOY_CONS"
```

Remarks

The variable can be set to the following values:

- "0": Left and right Joy-Con controllers will not be combined and each will be a mini-gamepad.
- "1": Left and right Joy-Con controllers will be combined into a single controller. (default)

This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_GAMECUBE

A variable controlling whether the HIDAPI driver for Nintendo GameCube controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_GAMECUBE "SDL_JOYSTICK_HIDAPI_GAMECUBE"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_GAMECUBE_RUMBLE_BRAKE

A variable controlling whether rumble is used to implement the GameCube controller's 3 rumble modes, Stop(0), Rumble(1), and StopHard(2).

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_GAMECUBE_RUMBLE_BRAKE
    "SDL_JOYSTICK_HIDAPI_GAMECUBE_RUMBLE_BRAKE"
```

Remarks

This is useful for applications that need full compatibility for things like ADSR envelopes. - Stop is implemented by setting low_frequency_rumble to 0 and high_frequency_rumble ¿0 - Rumble is both at any arbitrary value - StopHard is implemented by setting both low_frequency_rumble and high_frequency_rumble to 0

The variable can be set to the following values:

- "0": Normal rumble behavior is behavior is used. (default)
- "1": Proper GameCube controller rumble behavior is used.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_JOY_CONS

A variable controlling whether the HIDAPI driver for Nintendo Switch Joy-Cons should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_JOY_CONS "SDL_JOYSTICK_HIDAPI_JOY_CONS"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_JOYCON_HOME_LED

A variable controlling whether the Home button LED should be turned on when a Nintendo Switch Joy-Con controller is opened.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_JOYCON_HOME_LED
    "SDL_JOYSTICK_HIDAPI_JOYCON_HOME_LED"
```

Remarks

The variable can be set to the following values:

- "0": home button LED is turned off
- "1": home button LED is turned on

By default the Home button LED state is not changed. This hint can also be set to a floating point value between 0.0 and 1.0 which controls the brightness of the Home button LED.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_LUNA

A variable controlling whether the HIDAPI driver for Amazon Luna controllers connected via Bluetooth should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_LUNA "SDL_JOYSTICK_HIDAPI_LUNA"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_NINTENDO_CLASSIC

A variable controlling whether the HIDAPI driver for Nintendo Online classic controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_NINTENDO_CLASSIC
    "SDL_JOYSTICK_HIDAPI_NINTENDO_CLASSIC"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_PS3

A variable controlling whether the HIDAPI driver for PS3 controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_PS3 "SDL_JOYSTICK_HIDAPI_PS3"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI on macOS, and "0" on other platforms.

For official Sony driver (sixaxis.sys) use SDL_HINT_JOYSTICK_HIDAPI_PS3_SIXAXIS_DRIVER. See https://github.com/ViGEm/DsHidMini for an alternative driver on Windows.

This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_PS3_SIXAXIS_DRIVER

A variable controlling whether the Sony driver (sixaxis.sys) for PS3 controllers (Sixaxis/DualShock 3) should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_PS3_SIXAXIS_DRIVER
    "SDL_JOYSTICK_HIDAPI_PS3_SIXAXIS_DRIVER"
```

Remarks

The variable can be set to the following values:

- "0": Sony driver (sixaxis.sys) is not used.
- "1": Sony driver (sixaxis.sys) is used.

The default value is 0.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_PS4

A variable controlling whether the HIDAPI driver for PS4 controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_PS4 "SDL_JOYSTICK_HIDAPI_PS4"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_PS4_RUMBLE

A variable controlling whether extended input reports should be used for PS4 controllers when using the HIDAPI driver.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_PS4_RUMBLE
    "SDL_JOYSTICK_HIDAPI_PS4_RUMBLE"
```

Remarks

The variable can be set to the following values:

- "0": extended reports are not enabled. (default)
- "1": extended reports are enabled.

Extended input reports allow rumble on Bluetooth PS4 controllers, but break DirectInput handling for applications that don't use SDL.

Once extended reports are enabled, they can not be disabled without power cycling the controller.

For compatibility with applications written for versions of SDL prior to the introduction of PS5 controller support, this value will also control the state of extended reports on PS5 controllers when the SDL_HINT_JOYSTICK_HIDAPI_PS5_RUMBLE hint is not explicitly set.

This hint can be enabled anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_PS5

A variable controlling whether the HIDAPI driver for PS5 controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_PS5 "SDL_JOYSTICK_HIDAPI_PS5"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_PS5_PLAYER_LED

A variable controlling whether the player LEDs should be lit to indicate which player is associated with a PS5 controller.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_PS5_PLAYER_LED
    "SDL_JOYSTICK_HIDAPI_PS5_PLAYER_LED"
```

Remarks

The variable can be set to the following values:

- "0": player LEDs are not enabled.
- "1": player LEDs are enabled. (default)

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_PS5_RUMBLE

A variable controlling whether extended input reports should be used for PS5 controllers when using the HIDAPI driver.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_PS5_RUMBLE
    "SDL_JOYSTICK_HIDAPI_PS5_RUMBLE"
```

Remarks

The variable can be set to the following values:

- "0": extended reports are not enabled. (default)
- "1": extended reports.

Extended input reports allow rumble on Bluetooth PS5 controllers, but break DirectInput handling for applications that don't use SDL.

Once extended reports are enabled, they can not be disabled without power cycling the controller.

For compatibility with applications written for versions of SDL prior to the introduction of PS5 controller support, this value defaults to the value of SDL_HINT_JOYSTICK_HIDAPI_PS4_RUMBLE.

This hint can be enabled anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_SHIELD

A variable controlling whether the HIDAPI driver for NVIDIA SHIELD controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_SHIELD "SDL_JOYSTICK_HIDAPI_SHIELD"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_STADIA

A variable controlling whether the HIDAPI driver for Google Stadia controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_STADIA "SDL_JOYSTICK_HIDAPI_STADIA"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_STEAM

A variable controlling whether the HIDAPI driver for Bluetooth Steam Controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_STEAM "SDL_JOYSTICK_HIDAPI_STEAM"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used. (default)
- "1": HIDAPI driver is used for Steam Controllers, which requires Bluetooth access and may prompt the user for permission on iOS and Android.

This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_STEAMDECK

A variable controlling whether the HIDAPI driver for the Steam Deck builtin controller should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_STEAMDECK
    "SDL_JOYSTICK_HIDAPI_STEAMDECK"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_SWITCH

A variable controlling whether the HIDAPI driver for Nintendo Switch controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_SWITCH "SDL_JOYSTICK_HIDAPI_SWITCH"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_SWITCH_HOME_LED

A variable controlling whether the Home button LED should be turned on when a Nintendo Switch Pro controller is opened.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_SWITCH_HOME_LED
    "SDL_JOYSTICK_HIDAPI_SWITCH_HOME_LED"
```

Remarks

The variable can be set to the following values:

- "0": Home button LED is turned off.
- "1": Home button LED is turned on.

By default the Home button LED state is not changed. This hint can also be set to a floating point value between 0.0 and 1.0 which controls the brightness of the Home button LED.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_SWITCH_PLAYER_LED

A variable controlling whether the player LEDs should be lit to indicate which player is associated with a Nintendo Switch controller.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_SWITCH_PLAYER_LED
    "SDL_JOYSTICK_HIDAPI_SWITCH_PLAYER_LED"
```

Remarks

The variable can be set to the following values:

- "0": Player LEDs are not enabled.
- "1": Player LEDs are enabled. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_VERTICAL_JOY_CONS

A variable controlling whether Nintendo Switch Joy-Con controllers will be in vertical mode when using the HIDAPI driver.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_VERTICAL_JOY_CONS
    "SDL_JOYSTICK_HIDAPI_VERTICAL_JOY_CONS"
```

Remarks

The variable can be set to the following values:

- "0": Left and right Joy-Con controllers will not be in vertical mode. (default)
- "1": Left and right Joy-Con controllers will be in vertical mode.

This hint should be set before opening a Joy-Con controller.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_WII

A variable controlling whether the HIDAPI driver for Nintendo Wii and Wii U controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_WII "SDL_JOYSTICK_HIDAPI_WII"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

This driver doesn't work with the dolphinbar, so the default is SDL_FALSE for now.

This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_WII_PLAYER_LED

A variable controlling whether the player LEDs should be lit to indicate which player is associated with a Wii controller.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_WII_PLAYER_LED
    "SDL_JOYSTICK_HIDAPI_WII_PLAYER_LED"
```

Remarks

The variable can be set to the following values:

- "0": Player LEDs are not enabled.
- "1": Player LEDs are enabled. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_XBOX

A variable controlling whether the HIDAPI driver for XBox controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_XBOX "SDL_JOYSTICK_HIDAPI_XBOX"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is "0" on Windows, otherwise the value of SDL_HINT_JOYSTICK_HIDAPI This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_XBOX_360

A variable controlling whether the HIDAPI driver for XBox 360 controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_XBOX_360 "SDL_JOYSTICK_HIDAPI_XBOX_360"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI_XBOX
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_XBOX_360_PLAYER_LED

A variable controlling whether the player LEDs should be lit to indicate which player is associated with an Xbox 360 controller.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_XBOX_360_PLAYER_LED
    "SDL_JOYSTICK_HIDAPI_XBOX_360_PLAYER_LED"
```

Remarks

The variable can be set to the following values:

- "0": Player LEDs are not enabled.
- "1": Player LEDs are enabled. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_XBOX_360_WIRELESS

A variable controlling whether the HIDAPI driver for XBox 360 wireless controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_XBOX_360_WIRELESS
    "SDL_JOYSTICK_HIDAPI_XBOX_360_WIRELESS"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI_XBOX_360
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_XBOX_ONE

A variable controlling whether the HIDAPI driver for XBox One controllers should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_XBOX_ONE "SDL_JOYSTICK_HIDAPI_XBOX_ONE"
```

Remarks

The variable can be set to the following values:

- "0": HIDAPI driver is not used.
- "1": HIDAPI driver is used.

The default is the value of SDL_HINT_JOYSTICK_HIDAPI_XBOX.
This hint should be set before enumerating controllers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_HIDAPI_XBOX_ONE_HOME_LED

A variable controlling whether the Home button LED should be turned on when an Xbox One controller is opened.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_HIDAPI_XBOX_ONE_HOME_LED
    "SDL_JOYSTICK_HIDAPI_XBOX_ONE_HOME_LED"
```

Remarks

The variable can be set to the following values:

- "0": Home button LED is turned off.
- "1": Home button LED is turned on.

By default the Home button LED state is not changed. This hint can also be set to a floating point value between 0.0 and 1.0 which controls the brightness of the Home button LED. The default brightness is 0.4.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_IOKIT

A variable controlling whether IOKit should be used for controller handling.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_IOKIT "SDL_JOYSTICK_IOKIT"
```

Remarks

The variable can be set to the following values:

- "0": IOKit is not used.
- "1": IOKit is used. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_LINUX_CLASSIC

A variable controlling whether to use the classic /dev/input/js *joystick interface or the newer /dev/input/event* joystick interface on Linux.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_LINUX_CLASSIC "SDL_JOYSTICK_LINUX_CLASSIC"
```

Remarks

The variable can be set to the following values:

- "0": Use /dev/input/event *(default)*
- *"1": Use /dev/input/js*

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_LINUX_DEADZONES

A variable controlling whether joysticks on Linux adhere to their HID-defined deadzones or return unfiltered values.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_LINUX_DEADZONES "SDL_JOYSTICK_LINUX_DEADZONES"
```

Remarks

The variable can be set to the following values:

- "0": Return unfiltered joystick axis values. (default)
- "1": Return axis values with deadzones taken into account.

This hint should be set before a controller is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_LINUX_DIGITAL_HATS

A variable controlling whether joysticks on Linux will always treat 'hat' axis inputs (ABS_HAT0X - ABS_HAT3Y) as 8-way digital hats without checking whether they may be analog.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_LINUX_DIGITAL_HATS
    "SDL_JOYSTICK_LINUX_DIGITAL_HATS"
```

Remarks

The variable can be set to the following values:

- "0": Only map hat axis inputs to digital hat outputs if the input axes appear to actually be digital. (default)
- "1": Always handle the input axes numbered ABS_HAT0X to ABS_HAT3Y as digital hats.

This hint should be set before a controller is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_LINUX_HAT_DEADZONES

A variable controlling whether digital hats on Linux will apply deadzones to their underlying input axes or use unfiltered values.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_LINUX_HAT_DEADZONES
    "SDL_JOYSTICK_LINUX_HAT_DEADZONES"
```

Remarks

The variable can be set to the following values:

- "0": Return digital hat values based on unfiltered input axis values.
- "1": Return digital hat values with deadzones on the input axes taken into account. (default)

This hint should be set before a controller is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_MFI

A variable controlling whether GCController should be used for controller handling.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_MFI "SDL_JOYSTICK_MFI"
```

Remarks

The variable can be set to the following values:

- "0": GCController is not used.
- "1": GCController is used. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_RAWINPUT

A variable controlling whether the RAWINPUT joystick drivers should be used for better handling XInput-capable devices.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_RAWINPUT "SDL_JOYSTICK_RAWINPUT"
```

Remarks

The variable can be set to the following values:

- "0": RAWINPUT drivers are not used.
- "1": RAWINPUT drivers are used. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_RAWINPUT_CORRELATE_XINPUT

A variable controlling whether the RAWINPUT driver should pull correlated data from XInput.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_RAWINPUT_CORRELATE_XINPUT
    "SDL_JOYSTICK_RAWINPUT_CORRELATE_XINPUT"
```

Remarks

The variable can be set to the following values:

- "0": RAWINPUT driver will only use data from raw input APIs.
- "1": RAWINPUT driver will also pull data from XInput and Windows.Gaming.Input, providing better trigger axes, guide button presses, and rumble support for Xbox controllers. (default)

This hint should be set before a gamepad is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_ROG_CHAKRAM

A variable controlling whether the ROG Chakram mice should show up as joysticks.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_ROG_CHAKRAM "SDL_JOYSTICK_ROG_CHAKRAM"
```

Remarks

The variable can be set to the following values:

- "0": ROG Chakram mice do not show up as joysticks. (default)
- "1": ROG Chakram mice show up as joysticks.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_THREAD

A variable controlling whether a separate thread should be used for handling joystick detection and raw input messages on Windows.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_THREAD "SDL_JOYSTICK_THREAD"
```

Remarks

The variable can be set to the following values:

- "0": A separate thread is not used. (default)
- "1": A separate thread is used for handling raw input messages.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_THROTTLE_DEVICES

A variable containing a list of throttle style controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_THROTTLE_DEVICES
    "SDL_JOYSTICK_THROTTLE_DEVICES"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_THROTTLE_DEVICES_EXCLUDED

A variable containing a list of devices that are not throttle style controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_THROTTLE_DEVICES_EXCLUDED
    "SDL_JOYSTICK_THROTTLE_DEVICES_EXCLUDED"
```

Remarks

This will override SDL_HINT_JOYSTICK_THROTTLE_DEVICES and the built in device list.

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_WGI

A variable controlling whether Windows.Gaming.Input should be used for controller handling.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_WGI "SDL_JOYSTICK_WGI"
```

Remarks

The variable can be set to the following values:

- "0": WGI is not used.
- "1": WGI is used. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_WHEEL_DEVICES

A variable containing a list of wheel style controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_WHEEL_DEVICES "SDL_JOYSTICK_WHEEL_DEVICES"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_WHEEL_DEVICES_EXCLUDED

A variable containing a list of devices that are not wheel style controllers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_WHEEL_DEVICES_EXCLUDED
    "SDL_JOYSTICK_WHEEL_DEVICES_EXCLUDED"
```

Remarks

This will override SDL_HINT_JOYSTICK_WHEEL_DEVICES and the built in device list.

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_JOYSTICK_ZERO_CENTERED_DEVICES

A variable containing a list of devices known to have all axes centered at zero.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_JOYSTICK_ZERO_CENTERED_DEVICES
    "SDL_JOYSTICK_ZERO_CENTERED_DEVICES"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint should be set before a controller is opened.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_KMSDRM_DEVICE_INDEX

A variable that controls what KMSDRM device to use.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_KMSDRM_DEVICE_INDEX "SDL_KMSDRM_DEVICE_INDEX"
```

Remarks

SDL might open something like "/dev/dri/cardNN" to access KMSDRM functionality, where "NN" is a device index number. SDL makes a guess at the best index to use (usually zero), but the app or user can set this hint to a number between 0 and 99 to force selection.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_KMSDRM_REQUIRE_DRM_MASTER

A variable that controls whether SDL requires DRM master access in order to initialize the KMSDRM video backend.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_KMSDRM_REQUIRE_DRM_MASTER
    "SDL_KMSDRM_REQUIRE_DRM_MASTER"
```

Remarks

The DRM subsystem has a concept of a "DRM master" which is a DRM client that has the ability to set planes, set cursor, etc. When SDL is DRM master, it can draw to the screen using the SDL rendering APIs. Without DRM master, SDL is still able to process input and query attributes of attached displays, but it cannot change display state or draw to the screen directly.

In some cases, it can be useful to have the KMSDRM backend even if it cannot be used for rendering. An app may want to use SDL for input processing while using another rendering API (such as an MMAL overlay on Raspberry Pi) or using its own code to render to DRM overlays that SDL doesn't support.

The variable can be set to the following values:

- "0": SDL will allow usage of the KMSDRM backend without DRM master.
- "1": SDL Will require DRM master to use the KMSDRM backend. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_LOGGING

A variable controlling the default SDL log levels.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_LOGGING "SDL_LOGGING"
```

Remarks

This variable is a comma separated set of category=level tokens that define the default logging levels for SDL applications.

The category can be a numeric category, one of "app", "error", "assert", "system", "audio", "video", "render", "input", "test", or for any unspecified category.

The level can be a numeric level, one of "verbose", "debug", "info", "warn", "error", "critical", or "quiet" to disable that category.

You can omit the category if you want to set the logging level for all categories.

If this hint isn't set, the default log levels are equivalent to: `app=info,assert=warn,test=verbose,*=error`

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MAC_BACKGROUND_APP

A variable controlling whether to force the application to become the foreground process when launched on macOS.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MAC_BACKGROUND_APP "SDL_MAC_BACKGROUND_APP"
```

Remarks

The variable can be set to the following values:

- "0": The application is brought to the foreground when launched. (default)
- "1": The application may remain in the background when launched.

This hint should be set before applicationDidFinishLaunching() is called.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MAC_CTRL_CLICK_EMULATE_RIGHT_CLICK

A variable that determines whether Ctrl+Click should generate a right-click event on macOS.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MAC_CTRL_CLICK_EMULATE_RIGHT_CLICK
    "SDL_MAC_CTRL_CLICK_EMULATE_RIGHT_CLICK"
```

Remarks

The variable can be set to the following values:

- "0": Ctrl+Click does not generate a right mouse button click event. (default)
- "1": Ctrl+Click generated a right mouse button click event.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MAC_OPENGL_ASYNC_DISPATCH

A variable controlling whether dispatching OpenGL context updates should block the dispatching thread until the main thread finishes processing on macOS.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MAC_OPENGL_ASYNC_DISPATCH
    "SDL_MAC_OPENGL_ASYNC_DISPATCH"
```

Remarks

The variable can be set to the following values:

- "0": Dispatching OpenGL context updates will block the dispatching thread until the main thread finishes processing. (default)
- "1": Dispatching OpenGL context updates will allow the dispatching thread to continue execution.

Generally you want the default, but if you have OpenGL code in a background thread on a Mac, and the main thread hangs because it's waiting for that background thread, but that background thread is also hanging because it's waiting for the main thread to do an update, this might fix your issue.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MAIN_CALLBACK_RATE

Request SDL_AppIterate() be called at a specific rate.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MAIN_CALLBACK_RATE "SDL_MAIN_CALLBACK_RATE"
```

Remarks

This number is in Hz, so "60" means try to iterate 60 times per second.

On some platforms, or if you are using SDL_main instead of SDL_AppIterate, this hint is ignored. When the hint can be used, it is allowed to be changed at any time.

This defaults to 60, and specifying NULL for the hint's value will restore the default.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_AUTO_CAPTURE

A variable controlling whether the mouse is captured while mouse buttons are pressed.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_AUTO_CAPTURE "SDL_MOUSE_AUTO_CAPTURE"
```

Remarks

The variable can be set to the following values:

- "0": The mouse is not captured while mouse buttons are pressed.
- "1": The mouse is captured while mouse buttons are pressed.

By default the mouse is captured while mouse buttons are pressed so if the mouse is dragged outside the window, the application continues to receive mouse events until the button is released.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_DOUBLE_CLICK_RADIUS

A variable setting the double click radius, in pixels.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_DOUBLE_CLICK_RADIUS
    "SDL_MOUSE_DOUBLE_CLICK_RADIUS"
```

Remarks

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_DOUBLE_CLICK_TIME

A variable setting the double click time, in milliseconds.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_DOUBLE_CLICK_TIME "SDL_MOUSE_DOUBLE_CLICK_TIME"
```

Remarks

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_FOCUS_CLICKTHROUGH

Allow mouse click events when clicking to focus an SDL window.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_FOCUS_CLICKTHROUGH "SDL_MOUSE_FOCUS_CLICKTHROUGH"
```

Remarks

The variable can be set to the following values:

- "0": Ignore mouse clicks that activate a window. (default)
- "1": Generate events for mouse clicks that activate a window.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_NORMAL_SPEED_SCALE

A variable setting the speed scale for mouse motion, in floating point, when the mouse is not in relative mode.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_NORMAL_SPEED_SCALE "SDL_MOUSE_NORMAL_SPEED_SCALE"
```

Remarks

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_RELATIVE_MODE_CENTER

A variable controlling whether relative mouse mode constrains the mouse to the center of the window.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_RELATIVE_MODE_CENTER
    "SDL_MOUSE_RELATIVE_MODE_CENTER"
```

Remarks

Constraining to the center of the window works better for FPS games and when the application is running over RDP. Constraining to the whole window works better for 2D games and increases the chance that the mouse will be in the correct position when using high DPI mice.

The variable can be set to the following values:

- "0": Relative mouse mode constrains the mouse to the window.
- "1": Relative mouse mode constrains the mouse to the center of the window. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_RELATIVE_MODE_WARP

A variable controlling whether relative mouse mode is implemented using mouse warping.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_RELATIVE_MODE_WARP "SDL_MOUSE_RELATIVE_MODE_WARP"
```

Remarks

The variable can be set to the following values:

- "0": Relative mouse mode uses raw input. (default)
- "1": Relative mouse mode uses mouse warping.

This hint can be set anytime relative mode is not currently enabled.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_RELATIVE_SPEED_SCALE

A variable setting the scale for mouse motion, in floating point, when the mouse is in relative mode.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_RELATIVE_SPEED_SCALE
    "SDL_MOUSE_RELATIVE_SPEED_SCALE"
```

Remarks

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_RELATIVE_SYSTEM_SCALE

A variable controlling whether the system mouse acceleration curve is used for relative mouse motion.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_RELATIVE_SYSTEM_SCALE
    "SDL_MOUSE_RELATIVE_SYSTEM_SCALE"
```

Remarks

The variable can be set to the following values:

- "0": Relative mouse motion will be unscaled. (default)
- "1": Relative mouse motion will be scaled using the system mouse acceleration curve.

If SDL_HINT_MOUSE_RELATIVE_SPEED_SCALE is set, that will override the system speed scale.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_RELATIVE_WARP_MOTION

A variable controlling whether a motion event should be generated for mouse warping in relative mode.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_RELATIVE_WARP_MOTION
    "SDL_MOUSE_RELATIVE_WARP_MOTION"
```

Remarks

The variable can be set to the following values:

- "0": Warping the mouse will not generate a motion event in relative mode
- "1": Warping the mouse will generate a motion event in relative mode

By default warping the mouse will not generate motion events in relative mode. This avoids the application having to filter out large relative motion due to warping.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_MOUSE_TOUCH_EVENTS

A variable controlling whether mouse events should generate synthetic touch events.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_MOUSE_TOUCH_EVENTS "SDL_MOUSE_TOUCH_EVENTS"
```

Remarks

The variable can be set to the following values:

- "0": Mouse events will not generate touch events. (default for desktop platforms)
- "1": Mouse events will generate touch events. (default for mobile platforms, such as Android and iOS)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_NO_SIGNAL_HANDLERS

Tell SDL not to catch the SIGINT or SIGTERM signals on POSIX platforms.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_NO_SIGNAL_HANDLERS "SDL_NO_SIGNAL_HANDLERS"
```

Remarks

The variable can be set to the following values:

- "0": SDL will install a SIGINT and SIGTERM handler, and when it catches a signal, convert it into an SDL_EVENT_QUIT event. (default)
- "1": SDL will not install a signal handler at all.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_OPENGL_ES_DRIVER

A variable controlling what driver to use for OpenGL ES contexts.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_OPENGL_ES_DRIVER "SDL_OPENGL_ES_DRIVER"
```

Remarks

On some platforms, currently Windows and X11, OpenGL drivers may support creating contexts with an OpenGL ES profile. By default SDL uses these profiles, when available, otherwise it attempts to load an OpenGL ES library, e.g. that provided by the ANGLE project. This variable controls whether SDL follows this default behaviour or will always load an OpenGL ES library.

Circumstances where this is useful include - Testing an app with a particular OpenGL ES implementation, e.g ANGLE, or emulator, e.g. those from ARM, Imagination or Qualcomm. - Resolving OpenGL ES function addresses at link time by linking with the OpenGL ES library instead of querying them at run time with SDL_GL_GetProcAddress().

Caution: for an application to work with the default behaviour across different OpenGL drivers it must query the OpenGL ES function addresses at run time using SDL_GL_GetProcAddress().

This variable is ignored on most platforms because OpenGL ES is native or not supported.

The variable can be set to the following values:

- "0": Use ES profile of OpenGL, if available. (default)
- "1": Load OpenGL ES library using the default library names.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_ORIENTATIONS

A variable controlling which orientations are allowed on iOS/Android.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_ORIENTATIONS "SDL_IOS_ORIENTATIONS"
```

Remarks

In some circumstances it is necessary to be able to explicitly control which UI orientations are allowed.

This variable is a space delimited list of the following values:

- "LandscapeLeft"
- "LandscapeRight"
- "Portrait"
- "PortraitUpsideDown"

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_PEN_DELAY_MOUSE_BUTTON

A variable controlling whether pen mouse button emulation triggers only when the pen touches the tablet surface.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_PEN_DELAY_MOUSE_BUTTON "SDL_PEN_DELAY_MOUSE_BUTTON"
```

Remarks

The variable can be set to the following values:

- "0": The pen reports mouse button press/release immediately when the pen button is pressed/released, and the pen tip touching the surface counts as left mouse button press.
- "1": Mouse button presses are sent when the pen first touches the tablet (analogously for releases). Not pressing a pen button simulates mouse button 1, pressing the first pen button simulates mouse button 2 etc.; it is not possible to report multiple buttons as pressed at the same time. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_PEN_NOT_MOUSE

A variable controlling whether to treat pen movement as separate from mouse movement.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_PEN_NOT_MOUSE "SDL_PEN_NOT_MOUSE"
```

Remarks

By default, pens report both SDL_MouseMotionEvent and SDL_PenMotionEvent updates (analogously for button presses). This hint allows decoupling mouse and pen updates.

This variable toggles between the following behaviour:

- "0": Pen acts as a mouse with mouse ID SDL_PEN_MOUSEID. (default) Use case: client application is not pen aware, user wants to use pen instead of mouse to interact.
- "1": Pen reports mouse clicks and movement events but does not update SDL-internal mouse state (buttons pressed, current mouse location). Use case: client application is not pen aware, user frequently alternates between pen and "real" mouse.
- "2": Pen reports no mouse events. Use case: pen-aware client application uses this hint to allow user to toggle between pen+mouse mode ("2") and pen-only mode ("1" or "0").

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_POLL_SENTINEL

A variable controlling the use of a sentinel event when polling the event queue.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_POLL_SENTINEL "SDL_POLL_SENTINEL"
```

Remarks

When polling for events, SDL_PumpEvents is used to gather new events from devices. If a device keeps producing new events between calls to SDL_PumpEvents, a poll loop will become stuck until the new events stop. This is most noticeable when moving a high frequency mouse.

The variable can be set to the following values:

- "0": Disable poll sentinels.
- "1": Enable poll sentinels. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_PREFERRED_LOCALES

Override for SDL_GetPreferredLocales().

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_PREFERRED_LOCALES "SDL_PREFERRED_LOCALES"
```

Remarks

If set, this will be favored over anything the OS might report for the user's preferred locales. Changing this hint at runtime will not generate a SDL_EVENT_LOCALE_CHANGED event (but if you can change the hint, you can push your own event, if you want).

The format of this hint is a comma-separated list of language and locale, combined with an underscore, as is a common format: "en_GB". Locale is optional: "en". So you might have a list like this: "en_GB,jp,es_PT"

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_QUIT_ON_LAST_WINDOW_CLOSE

A variable that decides whether to send SDL_EVENT_QUIT when closing the last window.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_QUIT_ON_LAST_WINDOW_CLOSE
    "SDL_QUIT_ON_LAST_WINDOW_CLOSE"
```

Remarks

The variable can be set to the following values:

- "0": SDL will not send an SDL_EVENT_QUIT event when the last window is requesting to close. Note that in this case, there are still other legitimate reasons one might get an SDL_EVENT_QUIT event: choosing "Quit" from the macOS menu bar, sending a SIGINT (ctrl-c) on Unix, etc.
- "1": SDL will send a quit event when the last window is requesting to close. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RENDER_DIRECT3D11_DEBUG

A variable controlling whether to enable Direct3D 11+'s Debug Layer.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RENDER_DIRECT3D11_DEBUG "SDL_RENDER_DIRECT3D11_DEBUG"
```

Remarks

This variable does not have any effect on the Direct3D 9 based renderer.

The variable can be set to the following values:

- "0": Disable Debug Layer use. (default)
- "1": Enable Debug Layer use.

This hint should be set before creating a renderer.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RENDER_DIRECT3D_THREADSAFE

A variable controlling whether the Direct3D device is initialized for thread-safe operations.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RENDER_DIRECT3D_THREADSAFE
    "SDL_RENDER_DIRECT3D_THREADSAFE"
```

Remarks

The variable can be set to the following values:

- "0": Thread-safety is not enabled. (default)
- "1": Thread-safety is enabled.

This hint should be set before creating a renderer.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RENDER_DRIVER

A variable specifying which render driver to use.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RENDER_DRIVER          "SDL_RENDER_DRIVER"
```

Remarks

If the application doesn't pick a specific renderer to use, this variable specifies the name of the preferred renderer. If the preferred renderer can't be initialized, the normal default renderer is used.

This variable is case insensitive and can be set to the following values:

- "direct3d"
- "direct3d11"
- "direct3d12"
- "opengl"
- "opengles2"
- "opengles"
- "metal"
- "vulkan"
- "software"

The default varies by platform, but it's the first one in the list that is available on the current platform.

This hint should be set before creating a renderer.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RENDER_LINE_METHOD

A variable controlling how the 2D render API renders lines.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RENDER_LINE_METHOD "SDL_RENDER_LINE_METHOD"
```

Remarks

The variable can be set to the following values:

- "0": Use the default line drawing method (Bresenham's line algorithm)
- "1": Use the driver point API using Bresenham's line algorithm (correct, draws many points)
- "2": Use the driver line API (occasionally misses line endpoints based on hardware driver quirks
- "3": Use the driver geometry API (correct, draws thicker diagonal lines)

This hint should be set before creating a renderer.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RENDER_METAL_PREFER_LOW_POWER_DEVICE

A variable controlling whether the Metal render driver select low power device over default one.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RENDER_METAL_PREFER_LOW_POWER_DEVICE
    "SDL_RENDER_METAL_PREFER_LOW_POWER_DEVICE"
```

Remarks

The variable can be set to the following values:

- "0": Use the prefered OS device. (default)
- "1": Select a low power device.

This hint should be set before creating a renderer.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RENDER_PS2_DYNAMIC_VSYNC

A variable controlling whether vsync is automatically disabled if doesn't reach enough FPS.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RENDER_PS2_DYNAMIC_VSYNC "SDL_RENDER_PS2_DYNAMIC_VSYNC"
```

Remarks

The variable can be set to the following values:

- "0": It will be using VSYNC as defined in the main flag. (default)
- "1": If VSYNC was previously enabled, then it will disable VSYNC if doesn't reach enough speed

This hint should be set before creating a renderer.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RENDER_VSYNC

A variable controlling whether updates to the SDL screen surface should be synchronized with the vertical refresh, to avoid tearing.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RENDER_VSYNC           "SDL_RENDER_VSYNC"
```

Remarks

This hint overrides the application preference when creating a renderer.

The variable can be set to the following values:

- "0": Disable vsync. (default)
- "1": Enable vsync.

This hint should be set before creating a renderer.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RENDER_VULKAN_DEBUG

A variable controlling whether to enable Vulkan Validation Layers.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RENDER_VULKAN_DEBUG "SDL_RENDER_VULKAN_DEBUG"
```

Remarks

This variable can be set to the following values:

- "0": Disable Validation Layer use
- "1": Enable Validation Layer use

By default, SDL does not use Vulkan Validation Layers.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RETURN_KEY_HIDES_IME

A variable to control whether the return key on the soft keyboard should hide the soft keyboard on Android and iOS.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RETURN_KEY_HIDES_IME "SDL_RETURN_KEY_HIDES_IME"
```

Remarks

The variable can be set to the following values:

- "0": The return key will be handled as a key event. (default)
- "1": The return key will hide the keyboard.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_ROG_GAMEPAD_MICE

A variable containing a list of ROG gamepad capable mice.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_ROG_GAMEPAD_MICE "SDL_ROG_GAMEPAD_MICE"
```

Remarks

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_ROG_GAMEPAD_MICE_EXCLUDED

A variable containing a list of devices that are not ROG gamepad capable mice.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_ROG_GAMEPAD_MICE_EXCLUDED
    "SDL_ROG_GAMEPAD_MICE_EXCLUDED"
```

Remarks

This will override SDL_HINT_ROG_GAMEPAD_MICE and the built in device list.

The format of the string is a comma separated list of USB VID/PID pairs in hexadecimal form, e.g. `0xAAAA/0xBBBB,0xCCCC/0xDDDD`

The variable can also take the form of "@file", in which case the named file will be loaded and interpreted as the value of the variable.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_RPI_VIDEO_LAYER

A variable controlling which Dispmanx layer to use on a Raspberry PI.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_RPI_VIDEO_LAYER       "SDL_RPI_VIDEO_LAYER"
```

Remarks

Also known as Z-order. The variable can take a negative or positive value. The default is 10000.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_SCREENSAVER_INHIBIT_ACTIVITY_NAME

Specify an "activity name" for screensaver inhibition.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_SCREENSAVER_INHIBIT_ACTIVITY_NAME
    "SDL_SCREENSAVER_INHIBIT_ACTIVITY_NAME"
```

Remarks

Some platforms, notably Linux desktops, list the applications which are inhibiting the screensaver or other power-saving features.

This hint lets you specify the "activity name" sent to the OS when SDL_DisableScreenSaver() is used (or the screensaver is automatically disabled). The contents of this hint are used when the screensaver is disabled. You should use a string that describes what your program is doing (and, therefore, why the screensaver is disabled). For example, "Playing a game" or "Watching a video".

Setting this to "" or leaving it unset will have SDL use a reasonable default: "Playing a game" or something similar.

This hint should be set before calling SDL_DisableScreenSaver()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_SHUTDOWN_DBUS_ON_QUIT

A variable controlling whether SDL calls dbus_shutdown() on quit.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_SHUTDOWN_DBUS_ON_QUIT "SDL_SHUTDOWN_DBUS_ON_QUIT"
```

Remarks

This is useful as a debug tool to validate memory leaks, but shouldn't ever be set in production applications, as other libraries used by the application might use dbus under the hood and this cause cause crashes if they continue after SDL_Quit().

The variable can be set to the following values:

- "0": SDL will not call dbus_shutdown() on quit. (default)
- "1": SDL will call dbus_shutdown() on quit.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_STORAGE_TITLE_DRIVER

A variable that specifies a backend to use for title storage.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_STORAGE_TITLE_DRIVER "SDL_STORAGE_TITLE_DRIVER"
```

Remarks

By default, SDL will try all available storage backends in a reasonable order until it finds one that can work, but this hint allows the app or user to force a specific target, such as "pc" if, say, you are on Steam but want to avoid SteamRemoteStorage for title data.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_STORAGE_USER_DRIVER

A variable that specifies a backend to use for user storage.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_STORAGE_USER_DRIVER "SDL_STORAGE_USER_DRIVER"
```

Remarks

By default, SDL will try all available storage backends in a reasonable order until it finds one that can work, but this hint allows the app or user to force a specific target, such as "pc" if, say, you are on Steam but want to avoid SteamRemoteStorage for user data.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_THREAD_FORCE_REALTIME_TIME_CRITICAL

Specifies whether SDL_THREAD_PRIORITY_TIME_CRITICAL should be treated as realtime.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_THREAD_FORCE_REALTIME_TIME_CRITICAL
    "SDL_THREAD_FORCE_REALTIME_TIME_CRITICAL"
```

Remarks

On some platforms, like Linux, a realtime priority thread may be subject to restrictions that require special handling by the application. This hint exists to let SDL know that the app is prepared to handle said restrictions.

On Linux, SDL will apply the following configuration to any thread that becomes realtime:

- The SCHED_RESET_ON_FORK bit will be set on the scheduling policy,
- An RLIMIT_RTTIME budget will be configured to the rtkit specified limit.
- Exceeding this limit will result in the kernel sending SIGKILL to the app, refer to the man pages for more information.

The variable can be set to the following values:

- "0": default platform specific behaviour
- "1": Force SDL_THREAD_PRIORITY_TIME_CRITICAL to a realtime scheduling policy

This hint should be set before calling SDL_SetThreadPriority()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_THREAD_PRIORITY_POLICY

A string specifying additional information to use with SDL_SetThreadPriority.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_THREAD_PRIORITY_POLICY    "SDL_THREAD_PRIORITY_POLICY"
```

Remarks

By default SDL_SetThreadPriority will make appropriate system changes in order to apply a thread priority. For example on systems using pthreads the scheduler policy is changed automatically to a policy that works well with a given priority. Code which has specific requirements can override SDL's default behavior with this hint.

pthread hint values are "current", "other", "fifo" and "rr". Currently no other platform hint values are defined but may be in the future.

On Linux, the kernel may send SIGKILL to realtime tasks which exceed the distro configured execution budget for rtkit. This budget can be queried through RLIMIT_RTTIME after calling SDL_SetThreadPriority().

This hint should be set before calling SDL_SetThreadPriority()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_TIMER_RESOLUTION

A variable that controls the timer resolution, in milliseconds.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_TIMER_RESOLUTION "SDL_TIMER_RESOLUTION"
```

Remarks

The higher resolution the timer, the more frequently the CPU services timer interrupts, and the more precise delays are, but this takes up power and CPU time. This hint is only used on Windows.

See this blog post for more information: http://randomascii.wordpress.com/2013/07/08/windows-timer-resolution-megawatts-wasted/

The default value is "1".

If this variable is set to "0", the system timer resolution is not set.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_TOUCH_MOUSE_EVENTS

A variable controlling whether touch events should generate synthetic mouse events.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_TOUCH_MOUSE_EVENTS "SDL_TOUCH_MOUSE_EVENTS"
```

Remarks

The variable can be set to the following values:

- "0": Touch events will not generate mouse events.
- "1": Touch events will generate mouse events. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_TRACKPAD_IS_TOUCH_ONLY

A variable controlling whether trackpads should be treated as touch devices.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_TRACKPAD_IS_TOUCH_ONLY "SDL_TRACKPAD_IS_TOUCH_ONLY"
```

Remarks

On macOS (and possibly other platforms in the future), SDL will report touches on a trackpad as mouse input, which is generally what users expect from this device; however, these are often actually full multitouch-capable touch devices, so it might be preferable to some apps to treat them as such.

The variable can be set to the following values:

- "0": Trackpad will send mouse events. (default)
- "1": Trackpad will send touch events.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_TV_REMOTE_AS_JOYSTICK

A variable controlling whether the Android / tvOS remotes should be listed as joystick devices, instead of sending keyboard events.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_TV_REMOTE_AS_JOYSTICK "SDL_TV_REMOTE_AS_JOYSTICK"
```

Remarks

The variable can be set to the following values:

- "0": Remotes send enter/escape/arrow key events.
- "1": Remotes are available as 2 axis, 2 button joysticks. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_ALLOW_SCREENSAVER

A variable controlling whether the screensaver is enabled.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_ALLOW_SCREENSAVER "SDL_VIDEO_ALLOW_SCREENSAVER"
```

Remarks

The variable can be set to the following values:

- "0": Disable screensaver. (default)
- "1": Enable screensaver.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_DOUBLE_BUFFER

Tell the video driver that we only want a double buffer.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_DOUBLE_BUFFER  "SDL_VIDEO_DOUBLE_BUFFER"
```

Remarks

By default, most lowlevel 2D APIs will use a triple buffer scheme that wastes no CPU time on waiting for vsync after issuing a flip, but introduces a frame of latency. On the other hand, using a double buffer scheme instead is recommended for cases where low latency is an important factor because we save a whole frame of latency.

We do so by waiting for vsync immediately after issuing a flip, usually just after eglSwapBuffers call in the backend's *_SwapWindow function.

This hint is currently supported on the following drivers:

- Raspberry Pi (raspberrypi)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_DRIVER

A variable that specifies a video backend to use.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_DRIVER "SDL_VIDEO_DRIVER"
```

Remarks

By default, SDL will try all available video backends in a reasonable order until it finds one that can work, but this hint allows the app or user to force a specific target, such as "x11" if, say, you are on Wayland but want to try talking to the X server instead.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_EGL_ALLOW_GETDISPLAY_FALLBACK

If eglGetPlatformDisplay fails, fall back to calling eglGetDisplay.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_EGL_ALLOW_GETDISPLAY_FALLBACK
    "SDL_VIDEO_EGL_GETDISPLAY_FALLBACK"
```

Remarks

The variable can be set to one of the following values:

- "0": Do not fall back to eglGetDisplay.
- "1": Fall back to eglGetDisplay if eglGetPlatformDisplay fails. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_FORCE_EGL

A variable controlling whether the OpenGL context should be created with EGL.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_FORCE_EGL "SDL_VIDEO_FORCE_EGL"
```

Remarks

The variable can be set to the following values:

- "0": Use platform-specific GL context creation API (GLX, WGL, CGL, etc). (default)
- "1": Use EGL

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_MAC_FULLSCREEN_SPACES

A variable that specifies the policy for fullscreen Spaces on macOS.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_MAC_FULLSCREEN_SPACES
    "SDL_VIDEO_MAC_FULLSCREEN_SPACES"
```

Remarks

The variable can be set to the following values:

- "0": Disable Spaces support (FULLSCREEN_DESKTOP won't use them and SDL_WINDOW_RESIZABLE windows won't offer the "fullscreen" button on their titlebars).
- "1": Enable Spaces support (FULLSCREEN_DESKTOP will use them and SDL_WINDOW_RESIZABLE windows will offer the "fullscreen" button on their titlebars). (default)

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_MINIMIZE_ON_FOCUS_LOSS

A variable controlling whether fullscreen windows are minimized when they lose focus.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_MINIMIZE_ON_FOCUS_LOSS
    "SDL_VIDEO_MINIMIZE_ON_FOCUS_LOSS"
```

Remarks

The variable can be set to the following values:

- "0": Fullscreen windows will not be minimized when they lose focus. (default)
- "1": Fullscreen windows are minimized when they lose focus.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_SYNC_WINDOW_OPERATIONS

A variable controlling whether all window operations will block until complete.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_SYNC_WINDOW_OPERATIONS
    "SDL_VIDEO_SYNC_WINDOW_OPERATIONS"
```

Remarks

Window systems that run asynchronously may not have the results of window operations that resize or move the window applied immediately upon the return of the requesting function. Setting this hint will cause such operations to block after every call until the pending operation has completed. Setting this to '1' is the equivalent of calling SDL_SyncWindow() after every function call.

Be aware that amount of time spent blocking while waiting for window operations to complete can be quite lengthy, as animations may have to complete, which can take upwards of multiple seconds in some cases.

The variable can be set to the following values:

- "0": Window operations are non-blocking. (default)
- "1": Window operations will block until completed.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_WAYLAND_ALLOW_LIBDECOR

A variable controlling whether the libdecor Wayland backend is allowed to be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_WAYLAND_ALLOW_LIBDECOR
    "SDL_VIDEO_WAYLAND_ALLOW_LIBDECOR"
```

Remarks

libdecor is used over xdg-shell when xdg-decoration protocol is unavailable.

The variable can be set to the following values:

- "0": libdecor use is disabled.
- "1": libdecor use is enabled. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_WAYLAND_EMULATE_MOUSE_WARP

Enable or disable hidden mouse pointer warp emulation, needed by some older games.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_WAYLAND_EMULATE_MOUSE_WARP
    "SDL_VIDEO_WAYLAND_EMULATE_MOUSE_WARP"
```

Remarks

Wayland requires the pointer confinement protocol to warp the mouse, but that is just a hint that the compositor is free to ignore, and warping the the pointer to or from regions outside of the focused window is prohibited. When this hint is set and the pointer is hidden, SDL will emulate mouse warps using relative mouse mode. This is required for some older games (such as Source engine games), which warp the mouse to the centre of the screen rather than using relative mouse motion. Note that relative mouse mode may have different mouse acceleration behaviour than pointer warps.

The variable can be set to the following values:

- "0": Attempts to warp the mouse will be made, if the appropriate protocol is available.
- "1": Some mouse warps will be emulated by forcing relative mouse mode.

If not set, this is automatically enabled unless an application uses relative mouse mode directly.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_WAYLAND_MODE_EMULATION

A variable controlling whether video mode emulation is enabled under Wayland.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_WAYLAND_MODE_EMULATION
    "SDL_VIDEO_WAYLAND_MODE_EMULATION"
```

Remarks

When this hint is set, a standard set of emulated CVT video modes will be exposed for use by the application. If it is disabled, the only modes exposed will be the logical desktop size and, in the case of a scaled desktop, the native display resolution.

The variable can be set to the following values:

- "0": Video mode emulation is disabled.
- "1": Video mode emulation is enabled. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_WAYLAND_MODE_SCALING

A variable controlling how modes with a non-native aspect ratio are displayed under Wayland.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_WAYLAND_MODE_SCALING
    "SDL_VIDEO_WAYLAND_MODE_SCALING"
```

Remarks

When this hint is set, the requested scaling will be used when displaying fullscreen video modes that don't match the display's native aspect ratio. This is contingent on compositor viewport support.

The variable can be set to the following values:

- "aspect" - Video modes will be displayed scaled, in their proper aspect ratio, with black bars.
- "stretch" - Video modes will be scaled to fill the entire display. (default)
- "none" - Video modes will be displayed as 1:1 with no scaling.

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_WAYLAND_PREFER_LIBDECOR

A variable controlling whether the libdecor Wayland backend is preferred over native decrations.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_WAYLAND_PREFER_LIBDECOR
    "SDL_VIDEO_WAYLAND_PREFER_LIBDECOR"
```

Remarks

When this hint is set, libdecor will be used to provide window decorations, even if xdg-decoration is available. (Note that, by default, libdecor will use xdg-decoration itself if available).

The variable can be set to the following values:

- "0": libdecor is enabled only if server-side decorations are unavailable. (default)
- "1": libdecor is always enabled if available.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_WAYLAND_SCALE_TO_DISPLAY

A variable forcing non-DPI-aware Wayland windows to output at 1:1 scaling.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_WAYLAND_SCALE_TO_DISPLAY
    "SDL_VIDEO_WAYLAND_SCALE_TO_DISPLAY"
```

Remarks

When this hint is set, Wayland windows that are not flagged as being DPI-aware will be output with scaling designed to force 1:1 pixel mapping.

This is intended to allow legacy applications to be displayed without desktop scaling being applied, and has issues with certain display configurations, as this forces the window to behave in a way that Wayland desktops were not designed to accommodate:

- Rounding errors can result with odd window sizes and/or desktop scales.
- The window may be unusably small.
- The window may jump in size at times.
- The window may appear to be larger than the desktop size to the application.
- Possible loss of cursor precision.

New applications should be designed with proper DPI awareness handling instead of enabling this.

The variable can be set to the following values:

- "0": Windows will be scaled normally.
- "1": Windows will be forced to scale to achieve 1:1 output.

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_WIN_D3DCOMPILER

A variable specifying which shader compiler to preload when using the Chrome ANGLE binaries.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_WIN_D3DCOMPILER
    "SDL_VIDEO_WIN_D3DCOMPILER"
```

Remarks

SDL has EGL and OpenGL ES2 support on Windows via the ANGLE project. It can use two different sets of binaries, those compiled by the user from source or those provided by the Chrome browser. In the later case, these binaries require that SDL loads a DLL providing the shader compiler.

The variable can be set to the following values:

- "d3dcompiler_46.dll" - best for Vista or later. (default)
- "d3dcompiler_43.dll" - for XP support.
- "none" - do not load any library, useful if you compiled ANGLE from source and included the compiler in your binaries.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_X11_NET_WM_BYPASS_COMPOSITOR

A variable controlling whether the X11 _NET_WM_BYPASS_COMPOSITOR hint should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_X11_NET_WM_BYPASS_COMPOSITOR
    "SDL_VIDEO_X11_NET_WM_BYPASS_COMPOSITOR"
```

Remarks

The variable can be set to the following values:

- "0": Disable _NET_WM_BYPASS_COMPOSITOR.
- "1": Enable _NET_WM_BYPASS_COMPOSITOR. (default)

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_X11_NET_WM_PING

A variable controlling whether the X11 _NET_WM_PING protocol should be supported.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_X11_NET_WM_PING "SDL_VIDEO_X11_NET_WM_PING"
```

Remarks

By default SDL will use _NET_WM_PING, but for applications that know they will not always be able to respond to ping requests in a timely manner they can turn it off to avoid the window manager thinking the app is hung.

The variable can be set to the following values:

- "0": Disable _NET_WM_PING.
- "1": Enable _NET_WM_PING. (default)

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_X11_SCALING_FACTOR

A variable forcing the content scaling factor for X11 displays.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_X11_SCALING_FACTOR "SDL_VIDEO_X11_SCALING_FACTOR"
```

Remarks

The variable can be set to a floating point value in the range 1.0-10.0f

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_X11_WINDOW_VISUALID

A variable forcing the visual ID chosen for new X11 windows.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_X11_WINDOW_VISUALID
    "SDL_VIDEO_X11_WINDOW_VISUALID"
```

Remarks

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VIDEO_X11_XRANDR

A variable controlling whether the X11 XRandR extension should be used.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VIDEO_X11_XRANDR       "SDL_VIDEO_X11_XRANDR"
```

Remarks

The variable can be set to the following values:

- "0": Disable XRandR.
- "1": Enable XRandR. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_VITA_TOUCH_MOUSE_DEVICE

A variable controlling which touchpad should generate synthetic mouse events.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_VITA_TOUCH_MOUSE_DEVICE "SDL_VITA_TOUCH_MOUSE_DEVICE"
```

Remarks

The variable can be set to the following values:

- "0": Only front touchpad should generate mouse events. (default)
- "1": Only back touchpad should generate mouse events.
- "2": Both touchpads should generate mouse events.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WAVE_FACT_CHUNK

A variable controlling how the fact chunk affects the loading of a WAVE file.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WAVE_FACT_CHUNK "SDL_WAVE_FACT_CHUNK"
```

Remarks

The fact chunk stores information about the number of samples of a WAVE file. The Standards Update from Microsoft notes that this value can be used to 'determine the length of the data in seconds'. This is especially useful for compressed formats (for which this is a mandatory chunk) if they produce multiple sample frames per block and truncating the block is not allowed. The fact chunk can exactly specify how many sample frames there should be in this case.

Unfortunately, most application seem to ignore the fact chunk and so SDL ignores it by default as well.

The variable can be set to the following values:

- "truncate" - Use the number of samples to truncate the wave data if the fact chunk is present and valid.
- "strict" - Like "truncate", but raise an error if the fact chunk is invalid, not present for non-PCM formats, or if the data chunk doesn't have that many samples.
- "ignorezero" - Like "truncate", but ignore fact chunk if the number of samples is zero.
- "ignore" - Ignore fact chunk entirely. (default)

This hint should be set before calling SDL_LoadWAV() or SDL_LoadWAV_IO()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WAVE_RIFF_CHUNK_SIZE

A variable controlling how the size of the RIFF chunk affects the loading of a WAVE file.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WAVE_RIFF_CHUNK_SIZE "SDL_WAVE_RIFF_CHUNK_SIZE"
```

Remarks

The size of the RIFF chunk (which includes all the sub-chunks of the WAVE file) is not always reliable. In case the size is wrong, it's possible to just ignore it and step through the chunks until a fixed limit is reached.

Note that files that have trailing data unrelated to the WAVE file or corrupt files may slow down the loading process without a reliable boundary. By default, SDL stops after 10000 chunks to prevent wasting time. Use the environment variable SDL_WAVE_CHUNK_LIMIT to adjust this value.

The variable can be set to the following values:

- "force" - Always use the RIFF chunk size as a boundary for the chunk search.
- "ignorezero" - Like "force", but a zero size searches up to 4 GiB. (default)
- "ignore" - Ignore the RIFF chunk size and always search up to 4 GiB.
- "maximum" - Search for chunks until the end of file. (not recommended)

This hint should be set before calling SDL_LoadWAV() or SDL_LoadWAV_IO()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WAVE_TRUNCATION

A variable controlling how a truncated WAVE file is handled.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WAVE_TRUNCATION "SDL_WAVE_TRUNCATION"
```

Remarks

A WAVE file is considered truncated if any of the chunks are incomplete or the data chunk size is not a multiple of the block size. By default, SDL decodes until the first incomplete block, as most applications seem to do.

The variable can be set to the following values:

- "verystrict" - Raise an error if the file is truncated.
- "strict" - Like "verystrict", but the size of the RIFF chunk is ignored.
- "dropframe" - Decode until the first incomplete sample frame.
- "dropblock" - Decode until the first incomplete block. (default)

This hint should be set before calling SDL_LoadWAV() or SDL_LoadWAV_IO()

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOW_ACTIVATE_WHEN_RAISED

A variable controlling whether the window is activated when the SDL_RaiseWindow function is called.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOW_ACTIVATE_WHEN_RAISED
    "SDL_WINDOW_ACTIVATE_WHEN_RAISED"
```

Remarks

The variable can be set to the following values:

- "0": The window is not activated when the SDL_RaiseWindow function is called.
- "1": The window is activated when the SDL_RaiseWindow function is called. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOW_ACTIVATE_WHEN_SHOWN

A variable controlling whether the window is activated when the SDL_ShowWindow function is called.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOW_ACTIVATE_WHEN_SHOWN
    "SDL_WINDOW_ACTIVATE_WHEN_SHOWN"
```

Remarks

The variable can be set to the following values:

- "0": The window is not activated when the SDL_ShowWindow function is called.
- "1": The window is activated when the SDL_ShowWindow function is called. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOW_ALLOW_TOPMOST

If set to "0" then never set the top-most flag on an SDL Window even if the application requests it.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOW_ALLOW_TOPMOST "SDL_WINDOW_ALLOW_TOPMOST"
```

Remarks

This is a debugging aid for developers and not expected to be used by end users.

The variable can be set to the following values:

- "0": don't allow topmost
- "1": allow topmost (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOW_FRAME_USABLE_WHILE_CURSOR_H

A variable controlling whether the window frame and title bar are interactive when the cursor is hidden.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOW_FRAME_USABLE_WHILE_CURSOR_HIDDEN
    "SDL_WINDOW_FRAME_USABLE_WHILE_CURSOR_HIDDEN"
```

Remarks

The variable can be set to the following values:

- "0": The window frame is not interactive when the cursor is hidden (no move, resize, etc).
- "1": The window frame is interactive when the cursor is hidden. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOWS_CLOSE_ON_ALT_F4

A variable controlling whether SDL generates window-close events for Alt+F4 on Windows.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOWS_CLOSE_ON_ALT_F4 "SDL_WINDOWS_CLOSE_ON_ALT_F4"
```

Remarks

The variable can be set to the following values:

- "0": SDL will only do normal key handling for Alt+F4.
- "1": SDL will generate a window-close event when it sees Alt+F4. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOWS_ENABLE_MENU_MNEMONICS

A variable controlling whether menus can be opened with their keyboard shortcut (Alt+mnemonic).

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOWS_ENABLE_MENU_MNEMONICS
    "SDL_WINDOWS_ENABLE_MENU_MNEMONICS"
```

Remarks

If the mnemonics are enabled, then menus can be opened by pressing the Alt key and the corresponding mnemonic (for example, Alt+F opens the File menu). However, in case an invalid mnemonic is pressed, Windows makes an audible beep to convey that nothing happened. This is true even if the window has no menu at all!

Because most SDL applications don't have menus, and some want to use the Alt key for other purposes, SDL disables mnemonics (and the beeping) by default.

Note: This also affects keyboard events: with mnemonics enabled, when a menu is opened from the keyboard, you will not receive a KEYUP event for the mnemonic key, and *might* not receive one for Alt.

The variable can be set to the following values:

- "0": Alt+mnemonic does nothing, no beeping. (default)
- "1": Alt+mnemonic opens menus, invalid mnemonics produce a beep.

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOWS_ENABLE_MESSAGELOOP

A variable controlling whether the windows message loop is processed by SDL.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOWS_ENABLE_MESSAGELOOP
    "SDL_WINDOWS_ENABLE_MESSAGELOOP"
```

Remarks

The variable can be set to the following values:

- "0": The window message loop is not run.
- "1": The window message loop is processed in SDL_PumpEvents(). (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOWS_FORCE_MUTEX_CRITICAL_SECTIO

A variable controlling whether SDL uses Critical Sections for mutexes on Windows.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOWS_FORCE_MUTEX_CRITICAL_SECTIONS
    "SDL_WINDOWS_FORCE_MUTEX_CRITICAL_SECTIONS"
```

Remarks

On Windows 7 and newer, Slim Reader/Writer Locks are available. They offer better performance, allocate no kernel resources and use less memory. SDL will fall back to Critical Sections on older OS versions or if forced to by this hint.

The variable can be set to the following values:

- "0": Use SRW Locks when available, otherwise fall back to Critical Sections. (default)
- "1": Force the use of Critical Sections in all cases.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOWS_FORCE_SEMAPHORE_KERNEL

A variable controlling whether SDL uses Kernel Semaphores on Windows.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOWS_FORCE_SEMAPHORE_KERNEL
    "SDL_WINDOWS_FORCE_SEMAPHORE_KERNEL"
```

Remarks

Kernel Semaphores are inter-process and require a context switch on every interaction. On Windows 8 and newer, the WaitOnAddress API is available. Using that and atomics to implement semaphores increases performance. SDL will fall back to Kernel Objects on older OS versions or if forced to by this hint.

The variable can be set to the following values:

- "0": Use Atomics and WaitOnAddress API when available, otherwise fall back to Kernel Objects. (default)
- "1": Force the use of Kernel Objects in all cases.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOWS_INTRESOURCE_ICON

A variable to specify custom icon resource id from RC file on Windows platform.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOWS_INTRESOURCE_ICON
    "SDL_WINDOWS_INTRESOURCE_ICON"
```

Remarks

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOWS_RAW_KEYBOARD

A variable controlling whether raw keyboard events are used on Windows.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOWS_RAW_KEYBOARD "SDL_WINDOWS_RAW_KEYBOARD"
```

Remarks

The variable can be set to the following values:

- "0": The Windows message loop is used for keyboard events.
- "1": Low latency raw keyboard events are used. (default)

This hint can be set anytime.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINDOWS_USE_D3D9EX

A variable controlling whether SDL uses the D3D9Ex API introduced in Windows Vista, instead of normal D3D9.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINDOWS_USE_D3D9EX "SDL_WINDOWS_USE_D3D9EX"
```

Remarks

Direct3D 9Ex contains changes to state management that can eliminate device loss errors during scenarios like Alt+Tab or UAC prompts. D3D9Ex may require some changes to your application to cope with the new behavior, so this is disabled by default.

For more information on Direct3D 9Ex, see:

- https://docs.microsoft.com/en-us/windows/win32/direct3darticles/graphics-apis-in-windows-vista#direct3d-9ex
- https://docs.microsoft.com/en-us/windows/win32/direct3darticles/direct3d-9ex-improvements

The variable can be set to the following values:

- "0": Use the original Direct3D 9 API. (default)
- "1": Use the Direct3D 9Ex API on Vista and later (and fall back if D3D9Ex is unavailable)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINRT_HANDLE_BACK_BUTTON

A variable controlling whether back-button-press events on Windows Phone to be marked as handled.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINRT_HANDLE_BACK_BUTTON "SDL_WINRT_HANDLE_BACK_BUTTON"
```

Remarks

Windows Phone devices typically feature a Back button. When pressed, the OS will emit back-button-press events, which apps are expected to handle in an appropriate manner. If apps do not explicitly mark these events as 'Handled', then the OS will invoke its default behavior for unhandled back-button-press events, which on Windows Phone 8 and 8.1 is to terminate the app (and attempt to switch to the previous app, or to the device's home screen).

Setting the SDL_HINT_WINRT_HANDLE_BACK_BUTTON hint to "1" will cause SDL to mark back-button-press events as Handled, if and when one is sent to the app.

Internally, Windows Phone sends back button events as parameters to special back-button-press callback functions. Apps that need to respond to back-button-press events are expected to register one or more callback functions for such, shortly after being launched (during the app's initialization phase). After the back button is pressed, the OS will invoke these callbacks. If the app's callback(s) do not explicitly mark the event as handled by the time they return, or if the app never registers one of these callback, the OS will consider the event un-handled, and it will apply its default back button behavior (terminate the app).

SDL registers its own back-button-press callback with the Windows Phone OS. This callback will emit a pair of SDL key-press events (SDL_EVENT_KEY_DOWN and SDL_EVENT_KEY_UP), each with a scancode of SDL_SCANCODE_AC_BACK, after which it will check the contents of the hint, SDL_HINT_WINRT_HANDLE_BACK_BUTTON. If the hint's value is set to "1", the back button event's Handled property will get set to 'true'. If the hint's value is set to something else, or if it is unset, SDL will leave the event's Handled property alone. (By default, the OS sets this property to 'false', to note.)

SDL apps can either set SDL_HINT_WINRT_HANDLE_BACK_BUTTON well before a back button is pressed, or can set it in direct-response to a back button being pressed.

In order to get notified when a back button is pressed, SDL apps should register a callback function with SDL_AddEventWatch(), and have it listen for SDL_EVENT_KEY_DOWN events that have a scancode of SDL_SCANCODE_AC_BACK. (Alternatively, SDL_EVENT_KEY_UP events can be listened-for. Listening for either event type is suitable.) Any value of SDL_HINT_WINRT_HANDLE_BACK_BUTTON set by such a callback, will be applied to the OS' current back-button-press event.

More details on back button behavior in Windows Phone apps can be found at the following page, on Microsoft's developer site: http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj247550(v=vs.105).aspx

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINRT_PRIVACY_POLICY_LABEL

A variable specifying the label text for a WinRT app's privacy policy link.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINRT_PRIVACY_POLICY_LABEL
    "SDL_WINRT_PRIVACY_POLICY_LABEL"
```

Remarks

Network-enabled WinRT apps must include a privacy policy. On Windows 8, 8.1, and RT, Microsoft mandates that this policy be available via the Windows Settings charm. SDL provides code to add a link there, with its label text being set via the optional hint, SDL_HINT_WINRT_PRIVACY_POLICY_LABEL.

Please note that a privacy policy's contents are not set via this hint. A separate hint, SDL_HINT_WINRT_PRIVACY_POLICY_URL, is used to link to the actual text of the policy.

The contents of this hint should be encoded as a UTF8 string.

The default value is "Privacy Policy".

For additional information on linking to a privacy policy, see the documentation for SDL_HINT_WINRT_PRIVACY_POLICY_URL.

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_WINRT_PRIVACY_POLICY_URL

A variable specifying the URL to a WinRT app's privacy policy.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_WINRT_PRIVACY_POLICY_URL "SDL_WINRT_PRIVACY_POLICY_URL"
```

Remarks

All network-enabled WinRT apps must make a privacy policy available to its users. On Windows 8, 8.1, and RT, Microsoft mandates that this policy be be available in the Windows Settings charm, as accessed from within the app. SDL provides code to add a URL-based link there, which can point to the app's privacy policy.

To setup a URL to an app's privacy policy, set SDL_HINT_WINRT_PRIVACY_POLICY_URL before calling any SDL_Init() functions. The contents of the hint should be a valid URL. For example, "http://www.example.com".

The default value is "", which will prevent SDL from adding a privacy policy link to the Settings charm. This hint should only be set during app init.

The label text of an app's "Privacy Policy" link may be customized via another hint, SDL_HINT_WINRT_PRIVACY_POLICY_LABEL.

Please note that on Windows Phone, Microsoft does not provide standard UI for displaying a privacy policy link, and as such, SDL_HINT_WINRT_PRIVACY_POLICY_URL will not get used on that platform. Network-enabled phone apps should display their privacy policy through some other, in-app means.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_X11_FORCE_OVERRIDE_REDIRECT

A variable controlling whether X11 windows are marked as override-redirect.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_X11_FORCE_OVERRIDE_REDIRECT
    "SDL_X11_FORCE_OVERRIDE_REDIRECT"
```

Remarks

If set, this _might_ increase framerate at the expense of the desktop not working as expected. Override-redirect windows aren't noticed by the window manager at all.

You should probably only use this for fullscreen windows, and you probably shouldn't even use it for that. But it's here if you want to try!

The variable can be set to the following values:

- "0": Do not mark the window as override-redirect. (default)
- "1": Mark the window as override-redirect.

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_X11_WINDOW_TYPE

A variable specifying the type of an X11 window.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_X11_WINDOW_TYPE "SDL_X11_WINDOW_TYPE"
```

Remarks

During SDL_CreateWindow, SDL uses the _NET_WM_WINDOW_TYPE X11 property to report to the window manager the type of window it wants to create. This might be set to various things if SDL_WINDOW_TOOLTIP or SDL_WINDOW_POPUP_MENU, etc, were specified. For "normal" windows that haven't set a specific type, this hint can be used to specify a custom type. For example, a dock window might set this to "_NET_WM_WINDOW_TYPE_DOCK".

This hint should be set before creating a window.

Version

This hint is available since SDL 3.0.0.

SDL_HINT_XINPUT_ENABLED

A variable controlling whether XInput should be used for controller handling.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
#define SDL_HINT_XINPUT_ENABLED "SDL_XINPUT_ENABLED"
```

Remarks

The variable can be set to the following values:

- "0": XInput is not enabled.
- "1": XInput is enabled. (default)

This hint should be set before SDL is initialized.

Version

This hint is available since SDL 3.0.0.

SDL_JOYSTICK_AXIS_MAX

The largest value an SDL_Joystick's axis can report.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
#define SDL_JOYSTICK_AXIS_MAX 32767
```

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_JOYSTICK_AXIS_MIN

SDL_JOYSTICK_AXIS_MIN

The smallest value an SDL_Joystick's axis can report.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
#define SDL_JOYSTICK_AXIS_MIN -32768
```

Remarks

This is a negative number!

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_JOYSTICK_AXIS_MAX

SDL_MAIN_USE_CALLBACKS

Inform SDL to use the main callbacks instead of main.

Header File

Defined in SDL3/SDL_main.h

Syntax

```
#define SDL_MAIN_USE_CALLBACKS 1
```

Remarks

SDL does not define this macro, but will check if it is defined when including `SDL_main.h`. If defined, SDL will expect the app to provide several functions: SDL_AppInit, SDL_AppEvent, SDL_AppIterate, and SDL_AppQuit. The app should not provide a `main` function in this case, and doing so will likely cause the build to fail.

Please see README/main-functions, (or docs/README-main-functions.md in the source tree) for a more detailed explanation.

Version

This macro is used by the headers since SDL 3.0.0.

See Also

- SDL_AppInit
- SDL_AppEvent
- SDL_AppIterate
- SDL_AppQuit

SDL_MAX_SINT16

A signed 16-bit integer type.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_MAX_SINT16 ((Sint16)0x7FFF)    /* 32767 */
```

Version

This macro is available since SDL 3.0.0.

SDL_MAX_SINT32

A signed 32-bit integer type.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_MAX_SINT32 ((Sint32)0x7FFFFFFF) /* 2147483647 */
```

Version

This macro is available since SDL 3.0.0.

SDL_MAX_SINT64

A signed 64-bit integer type.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_MAX_SINT64 ((Sint64)0x7FFFFFFFFFFFFFFFll) /*
    9223372036854775807 */
```

Version

This macro is available since SDL 3.0.0.

SDL_MAX_SINT8

A signed 8-bit integer type.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_MAX_SINT8 ((Sint8)0x7F)        /* 127 */
```

Version

This macro is available since SDL 3.0.0.

SDL_MAX_TIME

SDL times are signed, 64-bit integers representing nanoseconds since the Unix epoch (Jan 1, 1970).

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_MAX_TIME SDL_MAX_SINT64
```

Remarks

They can be converted between POSIX time_t values with SDL_NS_TO_SECONDS() and SDL_SECONDS_TO_NS(), and between Windows FILETIME values with SDL_TimeToWindows() and SDL_TimeFromWindows().

Version

This macro is available since SDL 3.0.0.

SDL_MAX_UINT16

An unsigned 16-bit integer type.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_MAX_UINT16 ((Uint16)0xFFFF)     /* 65535 */
```

Version

This macro is available since SDL 3.0.0.

SDL_MAX_UINT32

An unsigned 32-bit integer type.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_MAX_UINT32 ((Uint32)0xFFFFFFFFu) /* 4294967295 */
```

Version

This macro is available since SDL 3.0.0.

SDL_MAX_UINT64

An unsigned 64-bit integer type.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_MAX_UINT64 ((Uint64)0xFFFFFFFFFFFFFFFFull) /*
    18446744073709551615 */
```

Version

This macro is available since SDL 3.0.0.

SDL_MAX_UINT8

An unsigned 8-bit integer type.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_MAX_UINT8 ((Uint8)0xFF)        /* 255 */
```

Version

This macro is available since SDL 3.0.0.

SDL_MIX_MAXVOLUME

Maximum volume allowed in calls to SDL_MixAudioFormat.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
#define SDL_MIX_MAXVOLUME 128
```

Version

This macro is available since SDL 3.0.0.

SDL_MUSTLOCK

Evaluates to true if the surface needs to be locked before access.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
#define SDL_MUSTLOCK(S) (((S)->flags & SDL_RLEACCEL) != 0)
```

Version

This macro is available since SDL 3.0.0.

SDL_MUTEX_TIMEDOUT

Synchronization functions return this value if they time out.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
#define SDL_MUTEX_TIMEDOUT 1
```

Remarks

Not all functions _can_ time out; some will block indefinitely.

Version

This macro is available since SDL 3.0.0.

SDL_PROP_GLOBAL_SYSTEM_DATE_FORMAT_NUMBER

The SDL_DateFormat to use as the preferred date display format for the current system locale.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
#define SDL_PROP_GLOBAL_SYSTEM_DATE_FORMAT_NUMBER "SDL.time.date_format"
```

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_PROP_GLOBAL_SYSTEM_TIME_FORMAT_NUMBER

SDL_PROP_GLOBAL_SYSTEM_TIME_FORMAT_NUMBER

The SDL_TimeFormat to use as the preferred time display format for the current system locale.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
#define SDL_PROP_GLOBAL_SYSTEM_TIME_FORMAT_NUMBER "SDL.time.time_format"
```

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_PROP_GLOBAL_SYSTEM_DATE_FORMAT_NUMBER

SDL_PROP_GLOBAL_VIDEO_WAYLAND_WL_DISPLAY_POINTER

The pointer to the global `wl_display` object used by the Wayland video backend.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
#define SDL_PROP_GLOBAL_VIDEO_WAYLAND_WL_DISPLAY_POINTER
    "SDL.video.wayland.wl_display"
```

Remarks

Can be set before the video subsystem is initialized to import an external `wl_display` object from an application or toolkit for use in SDL, or read after initialization to export the `wl_display` used by the Wayland video backend. Setting this property after the video subsystem has been initialized has no effect, and reading it when the video subsystem is uninitialized will either return the user provided value, if one was set prior to initialization, or NULL. See docs/README-wayland.md for more information.

SDL_REVISION

This macro is a string describing the source at a particular point in development.

Header File

Defined in SDL3/SDL_revision.h

Syntax

```
#define SDL_REVISION "Some arbitrary string decided at SDL build time"
```

Remarks

This string is often generated from revision control's state at build time.

This string can be quite complex and does not follow any standard. For example, it might be something like "SDL-prerelease-3.1.1-47-gf687e0732". It might also be user-defined at build time, so it's best to treat it as a clue in debugging forensics and not something the app will parse in any way.

Version

This macro is available since SDL 3.0.0.

SDL_SOFTWARE_RENDERER

The name of the software renderer.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
#define SDL_SOFTWARE_RENDERER "software"
```

Version

This macro is available since SDL 3.0.0.

SDL_STANDARD_GRAVITY

A constant to represent standard gravity for accelerometer sensors.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
#define SDL_STANDARD_GRAVITY 9.80665f
```

Remarks

The accelerometer returns the current acceleration in SI meters per second squared. This measurement includes the force of gravity, so a device at rest will have an value of SDL_STANDARD_GRAVITY away from the center of the earth, which is a positive Y value.

Version

This macro is available since SDL 3.0.0.

SDL_STRINGIFY_ARG

Macro useful for building other macros with strings in them.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_STRINGIFY_ARG(arg) #arg
```

Remarks

For example:

```
#define LOG_ERROR(X) OutputDebugString(SDL_STRINGIFY_ARG(__FUNCTION__)
    ": " X "\n")`
```

Version

This macro is available since SDL 3.0.0.

SDL_SwapBE16

Swap a 16-bit value from bigendian to native byte order.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
#define SDL_SwapBE16(x) SwapOnlyIfNecessary(x)
```

Return Value

Returns `x` in native byte order.

Remarks

If this is running on a bigendian system, `x` is returned unchanged.

This macro never references `x` more than once, avoiding side effects.

Version

This macro is available since SDL 3.0.0.

SDL_SwapBE32

Swap a 32-bit value from bigendian to native byte order.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
#define SDL_SwapBE32(x) SwapOnlyIfNecessary(x)
```

Return Value

Returns `x` in native byte order.

Remarks

If this is running on a bigendian system, `x` is returned unchanged.

This macro never references `x` more than once, avoiding side effects.

Version

This macro is available since SDL 3.0.0.

SDL_SwapBE64

Swap a 64-bit value from bigendian to native byte order.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
#define SDL_SwapBE64(x) SwapOnlyIfNecessary(x)
```

Return Value

Returns `x` in native byte order.

Remarks

If this is running on a bigendian system, `x` is returned unchanged.

This macro never references `x` more than once, avoiding side effects.

Version

This macro is available since SDL 3.0.0.

SDL_SwapFloatBE

Swap a floating point value from bigendian to native byte order.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
#define SDL_SwapFloatBE(x) SwapOnlyIfNecessary(x)
```

Return Value

Returns `x` in native byte order.

Remarks

If this is running on a bigendian system, `x` is returned unchanged.

This macro never references `x` more than once, avoiding side effects.

Version

This macro is available since SDL 3.0.0.

SDL_SwapFloatLE

Swap a floating point value from littleendian to native byte order.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
#define SDL_SwapFloatLE(x) SwapOnlyIfNecessary(x)
```

Return Value

Returns `x` in native byte order.

Remarks

If this is running on a littleendian system, `x` is returned unchanged.

This macro never references `x` more than once, avoiding side effects.

Version

This macro is available since SDL 3.0.0.

SDL_SwapLE16

Swap a 16-bit value from littleendian to native byte order.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
#define SDL_SwapLE16(x) SwapOnlyIfNecessary(x)
```

Return Value

Returns `x` in native byte order.

Remarks

If this is running on a littleendian system, `x` is returned unchanged.

This macro never references `x` more than once, avoiding side effects.

Version

This macro is available since SDL 3.0.0.

SDL_SwapLE32

Swap a 32-bit value from littleendian to native byte order.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
#define SDL_SwapLE32(x) SwapOnlyIfNecessary(x)
```

Return Value

Returns `x` in native byte order.

Remarks

If this is running on a littleendian system, `x` is returned unchanged.

This macro never references `x` more than once, avoiding side effects.

Version

This macro is available since SDL 3.0.0.

SDL_SwapLE64

Swap a 64-bit value from littleendian to native byte order.

Header File

Defined in SDL3/SDL_endian.h

Syntax

```
#define SDL_SwapLE64(x) SwapOnlyIfNecessary(x)
```

Return Value

Returns `x` in native byte order.

Remarks

If this is running on a littleendian system, `x` is returned unchanged.

This macro never references `x` more than once, avoiding side effects.

Version

This macro is available since SDL 3.0.0.

SDL_TriggerBreakpoint

Attempt to tell an attached debugger to pause.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
#define SDL_TriggerBreakpoint()
    TriggerABreakpointInAPlatformSpecificManner
```

Remarks

This allows an app to programmatically halt ("break") the debugger as if it had hit a breakpoint, allowing the developer to examine program state, etc.

This is a macro–not a function–so that the debugger breaks on the source code line that used SDL_TriggerBreakpoint and not in some random guts of SDL. SDL_assert uses this macro for the same reason.

If the program is not running under a debugger, SDL_TriggerBreakpoint will likely terminate the app, possibly without warning. If the current platform isn't supported (SDL doesn't know how to trigger a breakpoint), this macro does nothing.

Thread Safety

It is safe to call this function from any thread.

Version

This macro is available since SDL 3.0.0.

SDL_TRUE

A boolean true.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
#define SDL_TRUE 1
```

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_bool

SDL_VERSION

Macro to determine SDL version program was compiled against.

Header File

Defined in SDL3/SDL_version.h

Syntax

```
#define SDL_VERSION(x)            \
{                                 \
    (x)->major = SDL_MAJOR_VERSION; \
    (x)->minor = SDL_MINOR_VERSION; \
    (x)->patch = SDL_PATCHLEVEL;   \
}
```

Remarks

This macro fills in an SDL_Version structure with the version of the library you compiled against. This is determined by what header the compiler uses. Note that if you dynamically linked the library, you might have a slightly newer or older version at runtime. That version can be determined with SDL_GetVersion(), which, unlike SDL_VERSION(), is not a macro.

Version

This macro is available since SDL 3.0.0.

See Also

- SDL_Version
- SDL_GetVersion

SDL_VERSION_ATLEAST

This macro will evaluate to true if compiled with SDL at least X.Y.Z.

Header File

Defined in SDL3/SDL_version.h

Syntax

```
#define SDL_VERSION_ATLEAST(X, Y, Z) \
    (SDL_COMPILEDVERSION >= SDL_VERSIONNUM(X, Y, Z))
```

Version

This macro is available since SDL 3.0.0.

SDL_VERSIONNUM

This macro turns the version numbers into a numeric value.

Header File

Defined in SDL3/SDL_version.h

Syntax

```
#define SDL_VERSIONNUM(major, minor, patch) \
    ((major) << 24 | (minor) << 8 | (patch) << 0)
```

Remarks

(1,2,3) becomes 0x1000203.

Version

This macro is available since SDL 3.0.0.

SDL_WINDOWPOS_CENTERED_MASK

Used to indicate that the window position should be centered.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
#define SDL_WINDOWPOS_CENTERED_MASK 0x2FFF0000u
```

Version

This macro is available since SDL 3.0.0.

SDL_WINDOWPOS_UNDEFINED_MASK

Used to indicate that you don't care what the window position is.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
#define SDL_WINDOWPOS_UNDEFINED_MASK 0x1FFF0000u
```

Version

This macro is available since SDL 3.0.0.

Chapter 3

SDL Datatypes

SDL_AssertData

Information about an assertion failure.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
typedef struct SDL_AssertData
{
    SDL_bool always_ignore; /**< SDL_TRUE if app should always continue
        when assertion is triggered. */
    unsigned int trigger_count; /**< Number of times this assertion has
        been triggered. */
    const char *condition; /**< A string of this assert's test code. */
    const char *filename; /**< The source file where this assert lives.
        */
    int linenum; /**< The line in `filename` where this assert lives. */
    const char *function; /**< The name of the function where this
        assert lives. */
    const struct SDL_AssertData *next; /**< next item in the linked
        list. */
} SDL_AssertData;
```

Remarks

This structure is filled in with information about a triggered assertion, used by the assertion handler, then added to the assertion report. This is returned as a linked list from SDL_GetAssertionReport().

Version

This struct is available since SDL 3.0.0.

SDL_AssertionHandler

A callback that fires when an SDL assertion fails.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
typedef SDL_AssertState (SDLCALL *SDL_AssertionHandler)( const
    SDL_AssertData* data, void* userdata);
```

Function Parameters

data	a pointer to the SDL_AssertData structure corresponding to the current assertion
userdata	what was passed as `userdata` to SDL_SetAssertionHandler()

Return Value

Returns an SDL_AssertState value indicating how to handle the failure.

Version

This datatype is available since SDL 3.0.0.

SDL_AssertState

Possible outcomes from a triggered assertion.

Header File

Defined in SDL3/SDL_assert.h

Syntax

```
typedef enum SDL_AssertState
{
    SDL_ASSERTION_RETRY, /**< Retry the assert immediately. */
    SDL_ASSERTION_BREAK, /**< Make the debugger trigger a breakpoint. */
    SDL_ASSERTION_ABORT, /**< Terminate the program. */
    SDL_ASSERTION_IGNORE, /**< Ignore the assert. */
    SDL_ASSERTION_ALWAYS_IGNORE /**< Ignore the assert from now on. */
} SDL_AssertState;
```

Remarks

When an enabled assertion triggers, it may call the assertion handler (possibly one provided by the app via SDL_SetAssertionHandler), which will return one of these values, possibly after asking the user.

Then SDL will respond based on this outcome (loop around to retry the condition, try to break in a debugger, kill the program, or ignore the problem).

Version

This enum is available since SDL 3.0.0.

SDL_AtomicInt

A type representing an atomic integer value.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
typedef struct SDL_AtomicInt { int value; } SDL_AtomicInt;
```

Remarks

This can be used to manage a value that is synchronized across multiple CPUs without a race condition; when an app sets a value with SDL_AtomicSet all other threads, regardless of the CPU it is running on, will see that value when retrieved with SDL_AtomicGet, regardless of CPU caches, etc.

This is also useful for atomic compare-and-swap operations: a thread can change the value as long as its current value matches expectations. When done in a loop, one can guarantee data consistency across threads without a lock (but the usual warnings apply: if you don't know what you're doing, or you don't do it carefully, you can confidently cause any number of disasters with this, so in most cases, you _should_ use a mutex instead of this!).

This is a struct so people don't accidentally use numeric operations on it directly. You have to use SDL_Atomic* functions.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_AtomicCompareAndSwap
- SDL_AtomicGet
- SDL_AtomicSet
- SDL_AtomicAdd

SDL_AudioDeviceEvent

Audio device event structure (event.adevice.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_AudioDeviceEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_AUDIO_DEVICE_ADDED, or
        ::SDL_EVENT_AUDIO_DEVICE_REMOVED, or
        ::SDL_EVENT_AUDIO_DEVICE_FORMAT_CHANGED */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_AudioDeviceID which;    /**< SDL_AudioDeviceID for the device
        being added or removed or changing */
    Uint8 iscapture;  /**< zero if an output device, non-zero if a
        capture device. */
    Uint8 padding1;
    Uint8 padding2;
    Uint8 padding3;
} SDL_AudioDeviceEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_AudioDeviceID

SDL Audio Device instance IDs.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
typedef Uint32 SDL_AudioDeviceID;
```

Remarks

Zero is used to signify an invalid/null device.

Version

This datatype is available since SDL 3.0.0.

SDL_AudioFormat

Audio format flags.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
typedef Uint16 SDL_AudioFormat;

#define SDL_AUDIO_U8     0x0008 /**< Unsigned 8-bit samples */
#define SDL_AUDIO_S8     0x8008 /**< Signed 8-bit samples */

#define SDL_AUDIO_S16LE  0x8010 /**< Signed 16-bit samples */
#define SDL_AUDIO_S16BE  0x9010 /**< As above, but big-endian byte
    order */

#define SDL_AUDIO_S32LE  0x8020 /**< 32-bit integer samples */
#define SDL_AUDIO_S32BE  0x9020 /**< As above, but big-endian byte
    order */

#define SDL_AUDIO_F32LE  0x8120 /**< 32-bit floating point samples */
#define SDL_AUDIO_F32BE  0x9120 /**< As above, but big-endian byte
    order */

#if SDL_BYTEORDER == SDL_LIL_ENDIAN
#define SDL_AUDIO_S16  SDL_AUDIO_S16LE
#define SDL_AUDIO_S32  SDL_AUDIO_S32LE
#define SDL_AUDIO_F32  SDL_AUDIO_F32LE
#else
#define SDL_AUDIO_S16  SDL_AUDIO_S16BE
#define SDL_AUDIO_S32  SDL_AUDIO_S32BE
#define SDL_AUDIO_F32  SDL_AUDIO_F32BE
#endif
```

Remarks

These are what the 16 bits in SDL_AudioFormat currently mean... (Unspecified bits are always zero).

```
++-----------------------sample is signed if set
||
||       ++-----------sample is bigendian if set
||       ||
||       ||          ++---sample is float if set
```

```
||       ||          ||
||       ||          || +=--sample bit size--++
||       ||          || ||                   ||
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
```

There are macros to query these bits.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_AUDIO_BITSIZE
- SDL_AUDIO_BYTESIZE
- SDL_AUDIO_ISINT
- SDL_AUDIO_ISFLOAT
- SDL_AUDIO_ISBIGENDIAN
- SDL_AUDIO_ISLITTLEENDIAN
- SDL_AUDIO_ISSIGNED
- SDL_AUDIO_ISUNSIGNED

SDL_AudioPostmixCallback

A callback that fires when data is about to be fed to an audio device.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
typedef void (SDLCALL *SDL_AudioPostmixCallback)(void *userdata, const
    SDL_AudioSpec *spec, float *buffer, int buflen);
```

Function Parameters

userdata	a pointer provided by the app through SDL_SetAudioDevicePostmixCallback, for its own use.
spec	the current format of audio that is to be submitted to the audio device.
buffer	the buffer of audio samples to be submitted. The callback can inspect and/or modify this data.
buflen	the size of `buffer` in bytes.

Remarks

This is useful for accessing the final mix, perhaps for writing a visualizer or applying a final effect to the audio data before playback.

This callback should run as quickly as possible and not block for any significant time, as this callback delays submission of data to the audio device, which can cause audio playback problems.

The postmix callback _must_ be able to handle any audio data format specified in `spec`, which can change between callbacks if the audio device changed. However, this only covers frequency and channel count; data is always provided here in SDL_AUDIO_F32 format.

Thread Safety

This will run from a background thread owned by SDL. The application is responsible for locking resources the callback touches that need to be protected.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_SetAudioDevicePostmixCallback

SDL_AudioSpec

Format specifier for audio data.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
typedef struct SDL_AudioSpec
{
    SDL_AudioFormat format;  /**< Audio data format */
    int channels;            /**< Number of channels: 1 mono, 2 stereo,
        etc */
    int freq;                /**< sample rate: sample frames per second
        */
} SDL_AudioSpec;
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_AudioFormat

SDL_AudioStream

The opaque handle that represents an audio stream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
typedef struct SDL_AudioStream SDL_AudioStream;
```

Remarks

SDL_AudioStream is an audio conversion interface.

- It can handle resampling data in chunks without generating artifacts, when it doesn't have the complete buffer available.
- It can handle incoming data in any variable size.
- It can handle input/output format changes on the fly.
- You push data as you have it, and pull it when you need it
- It can also function as a basic audio data queue even if you just have sound that needs to pass from one place to another.
- You can hook callbacks up to them when more data is added or requested, to manage data on-the-fly.

Audio streams are the core of the SDL3 audio interface. You create one or more of them, bind them to an opened audio device, and feed data to them (or for recording, consume data from them).

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_CreateAudioStream

SDL_AudioStreamCallback

A callback that fires when data passes through an SDL_AudioStream.

Header File

Defined in SDL3/SDL_audio.h

Syntax

```
typedef void (SDLCALL *SDL_AudioStreamCallback)(void *userdata,
    SDL_AudioStream *stream, int additional_amount, int total_amount);
```

Function Parameters

stream	The SDL audio stream associated with this callback.
additional_amount	The amount of data, in bytes, that is needed right now.
total_amount	The total amount of data requested, in bytes, that is requested or available.
userdata	An opaque pointer provided by the app for their personal use.

Remarks

Apps can (optionally) register a callback with an audio stream that is called when data is added with SDL_PutAudioStreamData, or requested with SDL_GetAudioStreamData.

Two values are offered here: one is the amount of additional data needed to satisfy the immediate request (which might be zero if the stream already has enough data queued) and the other is the total amount being requested. In a Get call triggering a Put callback, these values can be different. In a Put call triggering a Get callback, these values are always the same.

Byte counts might be slightly overestimated due to buffering or resampling, and may change from call to call.

This callback is not required to do anything. Generally this is useful for adding/reading data on demand, and the app will often put/get data as appropriate, but the system goes on with the data currently available to it if this callback does nothing.

Thread Safety

This callbacks may run from any thread, so if you need to protect shared data, you should use SDL_LockAudioStream to serialize access; this lock will be held before your callback is called, so your callback does not need to manage the lock explicitly.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_SetAudioStreamGetCallback
- SDL_SetAudioStreamPutCallback

SDL_BlendFactor

The normalized factor used to multiply pixel components.

Header File

Defined in SDL3/SDL_blendmode.h

Syntax

```
typedef enum SDL_BlendFactor
{
    SDL_BLENDFACTOR_ZERO            = 0x1, /**< 0, 0, 0, 0 */
    SDL_BLENDFACTOR_ONE             = 0x2, /**< 1, 1, 1, 1 */
    SDL_BLENDFACTOR_SRC_COLOR       = 0x3, /**< srcR, srcG, srcB, srcA
        */
    SDL_BLENDFACTOR_ONE_MINUS_SRC_COLOR = 0x4, /**< 1-srcR, 1-srcG,
        1-srcB, 1-srcA */
    SDL_BLENDFACTOR_SRC_ALPHA       = 0x5, /**< srcA, srcA, srcA, srcA
        */
    SDL_BLENDFACTOR_ONE_MINUS_SRC_ALPHA = 0x6, /**< 1-srcA, 1-srcA,
        1-srcA, 1-srcA */
    SDL_BLENDFACTOR_DST_COLOR       = 0x7, /**< dstR, dstG, dstB, dstA
        */
    SDL_BLENDFACTOR_ONE_MINUS_DST_COLOR = 0x8, /**< 1-dstR, 1-dstG,
        1-dstB, 1-dstA */
    SDL_BLENDFACTOR_DST_ALPHA       = 0x9, /**< dstA, dstA, dstA, dstA
        */
    SDL_BLENDFACTOR_ONE_MINUS_DST_ALPHA = 0xA /**< 1-dstA, 1-dstA,
        1-dstA, 1-dstA */
} SDL_BlendFactor;
```

Remarks

The blend factors are multiplied with the pixels from a drawing operation (src) and the pixels from the render target (dst) before the blend operation. The comma-separated factors listed above are always applied in the component order red, green, blue, and alpha.

Version

This enum is available since SDL 3.0.0.

SDL_BlendMode

An enumeration of blend modes used in drawing operations.

Header File

Defined in SDL3/SDL_blendmode.h

Syntax

```
typedef enum SDL_BlendMode
{
    SDL_BLENDMODE_NONE = 0x00000000, /**< no blending
                                         dstRGBA = srcRGBA */
    SDL_BLENDMODE_BLEND = 0x00000001, /**< alpha blending
                                         dstRGB = (srcRGB * srcA) +
                                             (dstRGB * (1-srcA))
                                         dstA = srcA + (dstA *
                                             (1-srcA)) */
    SDL_BLENDMODE_ADD = 0x00000002,  /**< additive blending
                                         dstRGB = (srcRGB * srcA) +
                                             dstRGB
                                         dstA = dstA */
    SDL_BLENDMODE_MOD = 0x00000004,  /**< color modulate
                                         dstRGB = srcRGB * dstRGB
                                         dstA = dstA */
    SDL_BLENDMODE_MUL = 0x00000008,  /**< color multiply
                                         dstRGB = (srcRGB * dstRGB) +
                                             (dstRGB * (1-srcA))
                                         dstA = dstA */
    SDL_BLENDMODE_INVALID = 0x7FFFFFFF

    /* Additional custom blend modes can be returned by
        SDL_ComposeCustomBlendMode() */

} SDL_BlendMode;
```

Remarks

Note that additional values may be obtained from SDL_ComposeCustomBlendMode.

Version

This enum is available since SDL 3.0.0.

See Also

- SDL_ComposeCustomBlendMode

SDL_BlendOperation

The blend operation used when combining source and destination pixel components.

Header File

Defined in SDL3/SDL_blendmode.h

Syntax

```
typedef enum SDL_BlendOperation
{
    SDL_BLENDOPERATION_ADD          = 0x1, /**< dst + src: supported by
        all renderers */
    SDL_BLENDOPERATION_SUBTRACT     = 0x2, /**< src - dst : supported
        by D3D9, D3D11, OpenGL, OpenGLES */
    SDL_BLENDOPERATION_REV_SUBTRACT = 0x3, /**< dst - src : supported
        by D3D9, D3D11, OpenGL, OpenGLES */
    SDL_BLENDOPERATION_MINIMUM      = 0x4, /**< min(dst, src) :
        supported by D3D9, D3D11 */
    SDL_BLENDOPERATION_MAXIMUM      = 0x5  /**< max(dst, src) :
        supported by D3D9, D3D11 */
} SDL_BlendOperation;
```

Version

This enum is available since SDL 3.0.0.

SDL_BlitMap

An opaque type used in SDL_Surface.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
typedef struct SDL_BlitMap SDL_BlitMap;
```

Remarks

This is used by SDL to keep track of how blit operations should work internally; it is not for use directly by applications.

Version

This struct is available since SDL 3.0.0.

SDL_bool

A boolean type: true or false.

Header File

Defined in SDL3/SDL_stdinc.h

Syntax

```
typedef int SDL_bool;
```

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_TRUE
- SDL_FALSE

SDL_Camera

The opaque structure used to identify an opened SDL camera.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
struct SDL_Camera;
```

Version

This struct is available since SDL 3.0.0.

SDL_CameraDeviceEvent

Camera device event structure (event.cdevice.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_CameraDeviceEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_CAMERA_DEVICE_ADDED,
        ::SDL_EVENT_CAMERA_DEVICE_REMOVED,
        ::SDL_EVENT_CAMERA_DEVICE_APPROVED,
        ::SDL_EVENT_CAMERA_DEVICE_DENIED */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_CameraDeviceID which;    /**< SDL_CameraDeviceID for the device
        being added or removed or changing */
} SDL_CameraDeviceEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_CameraDeviceID

This is a unique ID for a camera device for the time it is connected to the system, and is never reused for the lifetime of the application.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
typedef Uint32 SDL_CameraDeviceID;
```

Remarks

If the device is disconnected and reconnected, it will get a new ID.

The ID value starts at 1 and increments from there. The value 0 is an invalid ID.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_GetCameraDevices

SDL_CameraPosition

The position of camera in relation to system device.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
typedef enum SDL_CameraPosition
{
    SDL_CAMERA_POSITION_UNKNOWN,
    SDL_CAMERA_POSITION_FRONT_FACING,
    SDL_CAMERA_POSITION_BACK_FACING
} SDL_CameraPosition;
```

Version

This enum is available since SDL 3.0.0.

See Also

- SDL_GetCameraDevicePosition

SDL_CameraSpec

The details of an output format for a camera device.

Header File

Defined in SDL3/SDL_camera.h

Syntax

```
typedef struct SDL_CameraSpec
{
    SDL_PixelFormatEnum format; /**< Frame format */
    int width;                  /**< Frame width */
    int height;                 /**< Frame height */
    int interval_numerator;     /**< Frame rate numerator ((dom / num) ==
        fps, (num / dom) == duration) */
    int interval_denominator; /**< Frame rate demoninator ((dom / num)
        == fps, (num / dom) == duration) */
} SDL_CameraSpec;
```

Remarks

Cameras often support multiple formats; each one will be encapsulated in this struct.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_GetCameraDeviceSupportedFormats
- SDL_GetCameraFormat

SDL_ChromaLocation

The chroma sample location.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef enum SDL_ChromaLocation
{
    SDL_CHROMA_LOCATION_NONE = 0, /**< RGB, no chroma sampling */
    SDL_CHROMA_LOCATION_LEFT = 1, /**< In MPEG-2, MPEG-4, and AVC, Cb
        and Cr are taken on midpoint of the left-edge of the 2x2 square.
        In other words, they have the same horizontal location as the
        top-left pixel, but is shifted one-half pixel down vertically. */
    SDL_CHROMA_LOCATION_CENTER = 2, /**< In JPEG/JFIF, H.261, and
        MPEG-1, Cb and Cr are taken at the center of the 2x2 square. In
        other words, they are offset one-half pixel to the right and
        one-half pixel down compared to the top-left pixel. */
    SDL_CHROMA_LOCATION_TOPLEFT = 3 /**< In HEVC for BT.2020 and BT.2100
        content (in particular on Blu-rays), Cb and Cr are sampled at
        the same location as the group's top-left Y pixel ("co-sited",
        "co-located"). */
} SDL_ChromaLocation;
```

Version

This enum is available since SDL 3.0.0.

SDL_ClipboardCleanupCallback

Callback function that will be called when the clipboard is cleared, or new data is set.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
typedef void (SDLCALL *SDL_ClipboardCleanupCallback)(void *userdata);
```

Function Parameters

userdata	A pointer to provided user data

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetClipboardData

SDL_ClipboardDataCallback

Callback function that will be called when data for the specified mime-type is requested by the OS.

Header File

Defined in SDL3/SDL_clipboard.h

Syntax

```
typedef const void *(SDLCALL *SDL_ClipboardDataCallback)(void *userdata,
    const char *mime_type, size_t *size);
```

Function Parameters

userdata	A pointer to provided user data
mime_type	The requested mime-type
size	A pointer filled in with the length of the returned data

Return Value

Returns a pointer to the data for the provided mime-type. Returning NULL or setting length to 0 will cause no data to be sent to the "receiver". It is up to the receiver to handle this. Essentially returning no data is more or less undefined behavior and may cause breakage in receiving applications. The returned data will not be freed so it needs to be retained and dealt with internally.

Remarks

The callback function is called with NULL as the mime_type when the clipboard is cleared or new data is set. The clipboard is automatically cleared in SDL_Quit().

Version

This function is available since SDL 3.0.0.

See Also

- SDL_SetClipboardData

SDL_ClipboardEvent

An event triggered when the clipboard contents have changed (event.clipboard.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_ClipboardEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_CLIPBOARD_UPDATE */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
} SDL_ClipboardEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_Color

A structure that represents a color as RGBA components.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef struct SDL_Color
{
   Uint8 r;
   Uint8 g;
   Uint8 b;
   Uint8 a;
} SDL_Color;
```

Remarks

The bits of this structure can be directly reinterpreted as an integer-packed color which uses the SDL_PIXELFORMAT_RGBA32 format (SDL_PIXELFORMAT_ABGR8888 on little-endian systems and SDL_PIXELFORMAT_RGBA8888 on big-endian systems).

Version

This struct is available since SDL 3.0.0.

SDL_ColorPrimaries

The color primaries, as described by https://www.itu.int/rec/T-REC-H.273-201612-S/en

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef enum SDL_ColorPrimaries
{
    SDL_COLOR_PRIMARIES_UNKNOWN = 0,
    SDL_COLOR_PRIMARIES_BT709 = 1,              /**< ITU-R BT.709-6 */
    SDL_COLOR_PRIMARIES_UNSPECIFIED = 2,
    SDL_COLOR_PRIMARIES_BT470M = 4,             /**< ITU-R BT.470-6
        System M */
    SDL_COLOR_PRIMARIES_BT470BG = 5,            /**< ITU-R BT.470-6
        System B, G / ITU-R BT.601-7 625 */
    SDL_COLOR_PRIMARIES_BT601 = 6,              /**< ITU-R BT.601-7 525 */
    SDL_COLOR_PRIMARIES_SMPTE240 = 7,           /**< SMPTE 240M,
        functionally the same as SDL_COLOR_PRIMARIES_BT601 */
    SDL_COLOR_PRIMARIES_GENERIC_FILM = 8,       /**< Generic film (color
        filters using Illuminant C) */
    SDL_COLOR_PRIMARIES_BT2020 = 9,             /**< ITU-R BT.2020-2 /
        ITU-R BT.2100-0 */
    SDL_COLOR_PRIMARIES_XYZ = 10,               /**< SMPTE ST 428-1 */
    SDL_COLOR_PRIMARIES_SMPTE431 = 11,          /**< SMPTE RP 431-2 */
    SDL_COLOR_PRIMARIES_SMPTE432 = 12,          /**< SMPTE EG 432-1 / DCI
        P3 */
    SDL_COLOR_PRIMARIES_EBU3213 = 22,           /**< EBU Tech. 3213-E */
    SDL_COLOR_PRIMARIES_CUSTOM = 31
} SDL_ColorPrimaries;
```

Version

This enum is available since SDL 3.0.0.

SDL_ColorRange

The color range, as described by https://www.itu.int/rec/R-REC-BT.2100-2-201807-I/en

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef enum SDL_ColorRange
{
    SDL_COLOR_RANGE_UNKNOWN = 0,
    SDL_COLOR_RANGE_LIMITED = 1, /**< Narrow range, e.g. 16-235 for
        8-bit RGB and luma, and 16-240 for 8-bit chroma */
    SDL_COLOR_RANGE_FULL = 2 /**< Full range, e.g. 0-255 for 8-bit RGB
        and luma, and 1-255 for 8-bit chroma */
} SDL_ColorRange;
```

Version

This enum is available since SDL 3.0.0.

SDL_ColorType

The color type

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef enum SDL_ColorType
{
    SDL_COLOR_TYPE_UNKNOWN = 0,
    SDL_COLOR_TYPE_RGB = 1,
    SDL_COLOR_TYPE_YCBCR = 2
} SDL_ColorType;
```

Version

This enum is available since SDL 3.0.0.

SDL_CommonEvent

Fields shared by every event

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_CommonEvent
{
    Uint32 type;      /**< Event type, shared with all events, Uint32 to
        cover user events which are not in the SDL_EventType enumeration
        */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
} SDL_CommonEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_Condition

A means to block multiple threads until a condition is satisfied.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
typedef struct SDL_Condition SDL_Condition;
```

Remarks

Condition variables, paired with an SDL_Mutex, let an app halt multiple threads until a condition has occurred, at which time the app can release one or all waiting threads.

Wikipedia has a thorough explanation of the concept:
https://en.wikipedia.org/wiki/Condition_variable

Version

This struct is available since SDL 3.0.0.

SDL_DateFormat

The preferred date format of the current system locale.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
typedef enum SDL_DateFormat
{
    SDL_DATE_FORMAT_YYYYMMDD = 0, /**< Year/Month/Day */
    SDL_DATE_FORMAT_DDMMYYYY = 1, /**< Day/Month/Year */
    SDL_DATE_FORMAT_MMDDYYYY = 2 /**< Month/Day/Year */
} SDL_DateFormat;
```

Version

This enum is available since SDL 3.0.0.

See Also

- SDL_PROP_GLOBAL_SYSTEM_DATE_FORMAT_NUMBER

SDL_DateTime

A structure holding a calendar date and time broken down into its components.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
typedef struct SDL_DateTime
{
    int year;               /**< Year */
    int month;              /**< Month [01-12] */
    int day;                /**< Day of the month [01-31] */
    int hour;               /**< Hour [0-23] */
    int minute;             /**< Minute [0-59] */
    int second;             /**< Seconds [0-60] */
    int nanosecond;         /**< Nanoseconds [0-999999999] */
    int day_of_week;        /**< Day of the week [0-6] (0 being Sunday)
        */
    int utc_offset;         /**< Seconds east of UTC */
} SDL_DateTime;
```

Version

This struct is available since SDL 3.0.0.

SDL_DialogFileCallback

Callback used by file dialog functions.

Header File

Defined in SDL3/SDL_dialog.h

Syntax

```
typedef void(SDLCALL *SDL_DialogFileCallback)(void *userdata, const char
    * const *filelist, int filter);
```

Remarks

The specific usage is described in each function.

If filelist is... - `NULL`, an error occured. Details can be obtained with SDL_GetError(). - A pointer to `NULL`, the user either didn't choose any file or canceled the dialog. - A pointer to non- `NULL`, the user chose one or more files. The argument is a null-terminated list of pointers to C strings, each containing a path.

The filelist argument does not need to be freed; it will automatically be freed when the callback returns.

The filter argument is the index of the filter that was selected, or one more than the size of the list (therefore the index of the terminating NULL entry) if no filter was selected, or -1 if the platform or method doesn't support fetching the selected filter.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_DialogFileFilter
- SDL_ShowOpenFileDialog
- SDL_ShowSaveFileDialog
- SDL_ShowOpenFolderDialog

SDL_DialogFileFilter

An entry for filters for file dialogs.

Header File

Defined in SDL3/SDL_dialog.h

Syntax

```
typedef struct SDL_DialogFileFilter
{
    const char *name;
    const char *pattern;
} SDL_DialogFileFilter;
```

Remarks

`name` is a user-readable label for the filter (for example, "Office document"). `pattern` is a semicolon-separated list of file extensions (for example, "doc;docx"). File extensions may only contain alphanumeric characters, hyphens, underscores and periods. Alternatively, the whole string can be a single asterisk ("*"), which serves as an "All files" filter.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_DialogFileCallback
- SDL_ShowOpenFileDialog
- SDL_ShowSaveFileDialog
- SDL_ShowOpenFolderDialog

SDL_DisplayEvent

Display state change event data (event.display.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_DisplayEvent
{
    SDL_EventType type; /**< ::SDL_DISPLAYEVENT_* */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_DisplayID displayID;/**< The associated display */
    Sint32 data1;     /**< event dependent data */
} SDL_DisplayEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_DisplayMode

The structure that defines a display mode.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef struct SDL_DisplayMode
{
    SDL_DisplayID displayID; /**< the display this mode is associated
        with */
    SDL_PixelFormatEnum format; /**< pixel format */
    int w;                   /**< width */
    int h;                   /**< height */
    float pixel_density;     /**< scale converting size to pixels (e.g.
        a 1920x1080 mode with 2.0 scale would have 3840x2160 pixels) */
    float refresh_rate;      /**< refresh rate (or zero for unspecified)
        */
    void *driverdata;        /**< driver-specific data, initialize to 0
        */
} SDL_DisplayMode;
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_GetFullscreenDisplayModes
- SDL_GetDesktopDisplayMode
- SDL_GetCurrentDisplayMode
- SDL_SetWindowFullscreenMode
- SDL_GetWindowFullscreenMode

SDL_DisplayOrientation

Display orientation values; the way a display is rotated.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef enum SDL_DisplayOrientation
{
    SDL_ORIENTATION_UNKNOWN,        /**< The display orientation can't
        be determined */
    SDL_ORIENTATION_LANDSCAPE,      /**< The display is in landscape
        mode, with the right side up, relative to portrait mode */
    SDL_ORIENTATION_LANDSCAPE_FLIPPED, /**< The display is in landscape
        mode, with the left side up, relative to portrait mode */
    SDL_ORIENTATION_PORTRAIT,       /**< The display is in portrait mode
        */
    SDL_ORIENTATION_PORTRAIT_FLIPPED /**< The display is in portrait
        mode, upside down */
} SDL_DisplayOrientation;
```

Version

This enum is available since SDL 3.0.0.

SDL_DropEvent

An event used to drop text or request a file open by the system (event.drop.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_DropEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_DROP_BEGIN or
        ::SDL_EVENT_DROP_FILE or ::SDL_EVENT_DROP_TEXT or
        ::SDL_EVENT_DROP_COMPLETE or ::SDL_EVENT_DROP_POSITION */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window that was dropped on, if any */
    float x;          /**< X coordinate, relative to window (not on
        begin) */
    float y;          /**< Y coordinate, relative to window (not on
        begin) */
    char *source;     /**< The source app that sent this drop event, or
        NULL if that isn't available */
    char *data;       /**< The text for SDL_EVENT_DROP_TEXT and the file
        name for SDL_EVENT_DROP_FILE, NULL for other events */
} SDL_DropEvent;
```

Remarks

The `data` is owned by SDL and should be copied if the application wants to hold onto it beyond the scope of handling this event. Do not free it!

Version

This struct is available since SDL 3.0.0.

SDL_EGLAttribArrayCallback

EGL attribute initialization callback types.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef SDL_EGLAttrib *(SDLCALL *SDL_EGLAttribArrayCallback)(void);
```

Version

This datatype is available since SDL 3.0.0.

SDL_EGLDisplay

Opaque EGL types.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef void *SDL_EGLDisplay;
```

Version

This datatype is available since SDL 3.0.0.

SDL_Event

The structure for all events in SDL.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef union SDL_Event
{
    Uint32 type;                        /**< Event type, shared with all
        events, Uint32 to cover user events which are not in the
        SDL_EventType enumeration */
    SDL_CommonEvent common;             /**< Common event data */
    SDL_DisplayEvent display;           /**< Display event data */
    SDL_WindowEvent window;             /**< Window event data */
    SDL_KeyboardDeviceEvent kdevice;    /**< Keyboard device change
        event data */
    SDL_KeyboardEvent key;              /**< Keyboard event data */
    SDL_TextEditingEvent edit;          /**< Text editing event data */
    SDL_TextInputEvent text;            /**< Text input event data */
    SDL_MouseDeviceEvent mdevice;       /**< Mouse device change event
        data */
    SDL_MouseMotionEvent motion;        /**< Mouse motion event data */
    SDL_MouseButtonEvent button;        /**< Mouse button event data */
    SDL_MouseWheelEvent wheel;          /**< Mouse wheel event data */
    SDL_JoyDeviceEvent jdevice;         /**< Joystick device change
        event data */
    SDL_JoyAxisEvent jaxis;             /**< Joystick axis event data */
    SDL_JoyBallEvent jball;             /**< Joystick ball event data */
    SDL_JoyHatEvent jhat;               /**< Joystick hat event data */
    SDL_JoyButtonEvent jbutton;         /**< Joystick button event data
        */
    SDL_JoyBatteryEvent jbattery;       /**< Joystick battery event data
        */
    SDL_GamepadDeviceEvent gdevice;     /**< Gamepad device event data */
    SDL_GamepadAxisEvent gaxis;         /**< Gamepad axis event data */
    SDL_GamepadButtonEvent gbutton;     /**< Gamepad button event data */
    SDL_GamepadTouchpadEvent gtouchpad; /**< Gamepad touchpad event data
        */
    SDL_GamepadSensorEvent gsensor;     /**< Gamepad sensor event data */
    SDL_AudioDeviceEvent adevice;       /**< Audio device event data */
    SDL_CameraDeviceEvent cdevice;      /**< Camera device event data */
    SDL_SensorEvent sensor;             /**< Sensor event data */
    SDL_QuitEvent quit;                 /**< Quit request event data */
    SDL_UserEvent user;                 /**< Custom event data */
```

```
    SDL_TouchFingerEvent tfinger;       /**< Touch finger event data */
    SDL_PenTipEvent ptip;               /**< Pen tip touching or leaving
        drawing surface */
    SDL_PenMotionEvent pmotion;         /**< Pen change in position,
        pressure, or angle */
    SDL_PenButtonEvent pbutton;         /**< Pen button press */
    SDL_DropEvent drop;                 /**< Drag and drop event data */
    SDL_ClipboardEvent clipboard;       /**< Clipboard event data */

    /* This is necessary for ABI compatibility between Visual C++ and
        GCC.
       Visual C++ will respect the push pack pragma and use 52 bytes
           (size of
       SDL_TextEditingEvent, the largest structure for 32-bit and 64-bit
       architectures) for this union, and GCC will use the alignment of
           the
       largest datatype within the union, which is 8 bytes on 64-bit
       architectures.

       So... we'll add padding to force the size to be the same for both.

       On architectures where pointers are 16 bytes, this needs rounding
           up to
       the next multiple of 16, 64, and on architectures where pointers
           are
       even larger the size of SDL_UserEvent will dominate as being 3
           pointers.
    */
    Uint8 padding[128];
} SDL_Event;
```

Version

This struct is available since SDL 3.0.0.

SDL_EventFilter

A function pointer used for callbacks that watch the event queue.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef int (SDLCALL *SDL_EventFilter)(void *userdata, SDL_Event *event);
```

Function Parameters

userdata	what was passed as `userdata` to SDL_SetEventFilter() or SDL_AddEventWatch, etc
event	the event that triggered the callback

Return Value

Returns 1 to permit event to be added to the queue, and 0 to disallow it. When used with SDL_AddEventWatch, the return value is ignored.

Thread Safety

SDL may call this callback at any time from any thread; the application is responsible for locking resources the callback touches that need to be protected.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_SetEventFilter
- SDL_AddEventWatch

SDL_EventType

The types of events that can be delivered.

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef enum SDL_EventType
{
    SDL_EVENT_FIRST   = 0,    /**< Unused (do not remove) */

    /* Application events */
    SDL_EVENT_QUIT        = 0x100, /**< User-requested quit */

    /* These application events have special meaning on iOS, see
        README-ios.md for details */
    SDL_EVENT_TERMINATING,     /**< The application is being terminated
        by the OS
                                Called on iOS in
                                    applicationWillTerminate()
                                Called on Android in onDestroy()
                            */
    SDL_EVENT_LOW_MEMORY,       /**< The application is low on memory,
        free memory if possible.
                                Called on iOS in
                                    applicationDidReceiveMemoryWarning()
                                Called on Android in onLowMemory()
                            */
    SDL_EVENT_WILL_ENTER_BACKGROUND, /**< The application is about to
        enter the background
                                Called on iOS in
                                    applicationWillResignActive()
                                Called on Android in onPause()
                            */
    SDL_EVENT_DID_ENTER_BACKGROUND, /**< The application did enter the
        background and may not get CPU for some time
                                Called on iOS in
                                    applicationDidEnterBackground()
                                Called on Android in onPause()
                            */
    SDL_EVENT_WILL_ENTER_FOREGROUND, /**< The application is about to
        enter the foreground
                                Called on iOS in
                                    applicationWillEnterForeground()
                                Called on Android in onResume()
```

```
                        */
SDL_EVENT_DID_ENTER_FOREGROUND, /**< The application is now
    interactive
                            Called on iOS in
                                applicationDidBecomeActive()
                            Called on Android in onResume()
                        */

SDL_EVENT_LOCALE_CHANGED, /**< The user's locale preferences have
    changed. */

SDL_EVENT_SYSTEM_THEME_CHANGED, /**< The system theme changed */

/* Display events */
/* 0x150 was SDL_DISPLAYEVENT, reserve the number for sdl2-compat */
SDL_EVENT_DISPLAY_ORIENTATION = 0x151, /**< Display orientation has
    changed to data1 */
SDL_EVENT_DISPLAY_ADDED,           /**< Display has been added to
    the system */
SDL_EVENT_DISPLAY_REMOVED,         /**< Display has been removed
    from the system */
SDL_EVENT_DISPLAY_MOVED,           /**< Display has changed position
    */
SDL_EVENT_DISPLAY_CONTENT_SCALE_CHANGED, /**< Display has changed
    content scale */
SDL_EVENT_DISPLAY_HDR_STATE_CHANGED, /**< Display HDR properties
    have changed */
SDL_EVENT_DISPLAY_FIRST = SDL_EVENT_DISPLAY_ORIENTATION,
SDL_EVENT_DISPLAY_LAST = SDL_EVENT_DISPLAY_HDR_STATE_CHANGED,

/* Window events */
/* 0x200 was SDL_WINDOWEVENT, reserve the number for sdl2-compat */
/* 0x201 was SDL_EVENT_SYSWM, reserve the number for sdl2-compat */
SDL_EVENT_WINDOW_SHOWN = 0x202,  /**< Window has been shown */
SDL_EVENT_WINDOW_HIDDEN,         /**< Window has been hidden */
SDL_EVENT_WINDOW_EXPOSED,        /**< Window has been exposed and
    should be redrawn */
SDL_EVENT_WINDOW_MOVED,          /**< Window has been moved to data1,
    data2 */
SDL_EVENT_WINDOW_RESIZED,        /**< Window has been resized to
    data1xdata2 */
SDL_EVENT_WINDOW_PIXEL_SIZE_CHANGED,/**< The pixel size of the
    window has changed to data1xdata2 */
SDL_EVENT_WINDOW_MINIMIZED,      /**< Window has been minimized */
SDL_EVENT_WINDOW_MAXIMIZED,      /**< Window has been maximized */
SDL_EVENT_WINDOW_RESTORED,       /**< Window has been restored to
    normal size and position */
SDL_EVENT_WINDOW_MOUSE_ENTER,    /**< Window has gained mouse focus */
SDL_EVENT_WINDOW_MOUSE_LEAVE,    /**< Window has lost mouse focus */
```

```
SDL_EVENT_WINDOW_FOCUS_GAINED,  /**< Window has gained keyboard
    focus */
SDL_EVENT_WINDOW_FOCUS_LOST,    /**< Window has lost keyboard focus
    */
SDL_EVENT_WINDOW_CLOSE_REQUESTED, /**< The window manager requests
    that the window be closed */
SDL_EVENT_WINDOW_TAKE_FOCUS,    /**< Window is being offered a focus
    (should SetWindowInputFocus() on itself or a subwindow, or
    ignore) */
SDL_EVENT_WINDOW_HIT_TEST,      /**< Window had a hit test that
    wasn't SDL_HITTEST_NORMAL */
SDL_EVENT_WINDOW_ICCPROF_CHANGED, /**< The ICC profile of the
    window's display has changed */
SDL_EVENT_WINDOW_DISPLAY_CHANGED, /**< Window has been moved to
    display data1 */
SDL_EVENT_WINDOW_DISPLAY_SCALE_CHANGED, /**< Window display scale
    has been changed */
SDL_EVENT_WINDOW_OCCLUDED,      /**< The window has been occluded */
SDL_EVENT_WINDOW_ENTER_FULLSCREEN, /**< The window has entered
    fullscreen mode */
SDL_EVENT_WINDOW_LEAVE_FULLSCREEN, /**< The window has left
    fullscreen mode */
SDL_EVENT_WINDOW_DESTROYED,     /**< The window with the associated
    ID is being or has been destroyed. If this message is being
    handled
                                    in an event watcher, the window
                                        handle is still valid and
                                        can still be used to
                                        retrieve any userdata
                                    associated with the window.
                                        Otherwise, the handle has
                                        already been destroyed and
                                        all resources
                                    associated with it are invalid
                                        */
SDL_EVENT_WINDOW_PEN_ENTER,     /**< Window has gained focus of the
    pressure-sensitive pen with ID "data1" */
SDL_EVENT_WINDOW_PEN_LEAVE,     /**< Window has lost focus of the
    pressure-sensitive pen with ID "data1" */
SDL_EVENT_WINDOW_FIRST = SDL_EVENT_WINDOW_SHOWN,
SDL_EVENT_WINDOW_LAST = SDL_EVENT_WINDOW_PEN_LEAVE,

/* Keyboard events */
SDL_EVENT_KEY_DOWN      = 0x300, /**< Key pressed */
SDL_EVENT_KEY_UP,               /**< Key released */
SDL_EVENT_TEXT_EDITING,         /**< Keyboard text editing
    (composition) */
SDL_EVENT_TEXT_INPUT,           /**< Keyboard text input */
SDL_EVENT_KEYMAP_CHANGED,       /**< Keymap changed due to a system
    event such as an
```

```
                                    input language or keyboard
                                        layout change. */
SDL_EVENT_KEYBOARD_ADDED,      /**< A new keyboard has been
    inserted into the system */
SDL_EVENT_KEYBOARD_REMOVED,    /**< A keyboard has been removed */

/* Mouse events */
SDL_EVENT_MOUSE_MOTION = 0x400, /**< Mouse moved */
SDL_EVENT_MOUSE_BUTTON_DOWN,   /**< Mouse button pressed */
SDL_EVENT_MOUSE_BUTTON_UP,     /**< Mouse button released */
SDL_EVENT_MOUSE_WHEEL,         /**< Mouse wheel motion */
SDL_EVENT_MOUSE_ADDED,         /**< A new mouse has been inserted
    into the system */
SDL_EVENT_MOUSE_REMOVED,       /**< A mouse has been removed */

/* Joystick events */
SDL_EVENT_JOYSTICK_AXIS_MOTION = 0x600, /**< Joystick axis motion */
SDL_EVENT_JOYSTICK_BALL_MOTION,      /**< Joystick trackball motion
    */
SDL_EVENT_JOYSTICK_HAT_MOTION,       /**< Joystick hat position
    change */
SDL_EVENT_JOYSTICK_BUTTON_DOWN,      /**< Joystick button pressed */
SDL_EVENT_JOYSTICK_BUTTON_UP,        /**< Joystick button released */
SDL_EVENT_JOYSTICK_ADDED,            /**< A new joystick has been
    inserted into the system */
SDL_EVENT_JOYSTICK_REMOVED,          /**< An opened joystick has
    been removed */
SDL_EVENT_JOYSTICK_BATTERY_UPDATED,  /**< Joystick battery level
    change */
SDL_EVENT_JOYSTICK_UPDATE_COMPLETE,  /**< Joystick update is
    complete */

/* Gamepad events */
SDL_EVENT_GAMEPAD_AXIS_MOTION = 0x650, /**< Gamepad axis motion */
SDL_EVENT_GAMEPAD_BUTTON_DOWN,      /**< Gamepad button pressed */
SDL_EVENT_GAMEPAD_BUTTON_UP,        /**< Gamepad button released */
SDL_EVENT_GAMEPAD_ADDED,           /**< A new gamepad has been
    inserted into the system */
SDL_EVENT_GAMEPAD_REMOVED,         /**< An opened gamepad has been
    removed */
SDL_EVENT_GAMEPAD_REMAPPED,        /**< The gamepad mapping was
    updated */
SDL_EVENT_GAMEPAD_TOUCHPAD_DOWN,    /**< Gamepad touchpad was
    touched */
SDL_EVENT_GAMEPAD_TOUCHPAD_MOTION,  /**< Gamepad touchpad finger was
    moved */
SDL_EVENT_GAMEPAD_TOUCHPAD_UP,      /**< Gamepad touchpad finger was
    lifted */
SDL_EVENT_GAMEPAD_SENSOR_UPDATE,    /**< Gamepad sensor was updated
    */
```

```
SDL_EVENT_GAMEPAD_UPDATE_COMPLETE,  /**< Gamepad update is complete
    */
SDL_EVENT_GAMEPAD_STEAM_HANDLE_UPDATED, /**< Gamepad Steam handle
    has changed */

/* Touch events */
SDL_EVENT_FINGER_DOWN   = 0x700,
SDL_EVENT_FINGER_UP,
SDL_EVENT_FINGER_MOTION,

/* 0x800, 0x801, and 0x802 were the Gesture events from SDL2. Do not
    reuse these values! sdl2-compat needs them! */

/* Clipboard events */
SDL_EVENT_CLIPBOARD_UPDATE = 0x900, /**< The clipboard or primary
    selection changed */

/* Drag and drop events */
SDL_EVENT_DROP_FILE     = 0x1000, /**< The system requests a file
    open */
SDL_EVENT_DROP_TEXT,             /**< text/plain drag-and-drop event
    */
SDL_EVENT_DROP_BEGIN,            /**< A new set of drops is
    beginning (NULL filename) */
SDL_EVENT_DROP_COMPLETE,         /**< Current set of drops is now
    complete (NULL filename) */
SDL_EVENT_DROP_POSITION,         /**< Position while moving over the
    window */

/* Audio hotplug events */
SDL_EVENT_AUDIO_DEVICE_ADDED = 0x1100, /**< A new audio device is
    available */
SDL_EVENT_AUDIO_DEVICE_REMOVED,     /**< An audio device has been
    removed. */
SDL_EVENT_AUDIO_DEVICE_FORMAT_CHANGED, /**< An audio device's format
    has been changed by the system. */

/* Sensor events */
SDL_EVENT_SENSOR_UPDATE = 0x1200, /**< A sensor was updated */

/* Pressure-sensitive pen events */
SDL_EVENT_PEN_DOWN    = 0x1300,    /**< Pressure-sensitive pen
    touched drawing surface */
SDL_EVENT_PEN_UP,                  /**< Pressure-sensitive pen
    stopped touching drawing surface */
SDL_EVENT_PEN_MOTION,              /**< Pressure-sensitive pen moved,
    or angle/pressure changed */
SDL_EVENT_PEN_BUTTON_DOWN,         /**< Pressure-sensitive pen button
    pressed */
```

```
    SDL_EVENT_PEN_BUTTON_UP,          /**< Pressure-sensitive pen button
        released */

    /* Camera hotplug events */
    SDL_EVENT_CAMERA_DEVICE_ADDED = 0x1400, /**< A new camera device is
        available */
    SDL_EVENT_CAMERA_DEVICE_REMOVED,     /**< A camera device has been
        removed. */
    SDL_EVENT_CAMERA_DEVICE_APPROVED,    /**< A camera device has been
        approved for use by the user. */
    SDL_EVENT_CAMERA_DEVICE_DENIED,      /**< A camera device has been
        denied for use by the user. */

    /* Render events */
    SDL_EVENT_RENDER_TARGETS_RESET = 0x2000, /**< The render targets
        have been reset and their contents need to be updated */
    SDL_EVENT_RENDER_DEVICE_RESET, /**< The device has been reset and
        all textures need to be recreated */

    /* Internal events */
    SDL_EVENT_POLL_SENTINEL = 0x7F00, /**< Signals the end of an event
        poll cycle */

    /** Events ::SDL_EVENT_USER through ::SDL_EVENT_LAST are for your
        use,
     *  and should be allocated with SDL_RegisterEvents()
     */
    SDL_EVENT_USER  = 0x8000,

    /**
     *  This last event is only for bounding internal arrays
     */
    SDL_EVENT_LAST  = 0xFFFF,

    /* This just makes sure the enum is the size of Uint32 */
    SDL_EVENT_ENUM_PADDING = 0x7FFFFFFF

} SDL_EventType;
```

Version

This enum is available since SDL 3.0.0.

SDL_FColor

The bits of this structure can be directly reinterpreted as a float-packed color which uses the SDL_PIXELFORMAT_RGBA128_FLOAT format

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef struct SDL_FColor
{
    float r;
    float g;
    float b;
    float a;
} SDL_FColor;
```

Version

This struct is available since SDL 3.0.0.

SDL_Finger

Data about a single finger in a multitouch event.

Header File

Defined in SDL3/SDL_touch.h

Syntax

```
typedef struct SDL_Finger
{
    SDL_FingerID id; /**< the finger ID */
    float x; /**< the x-axis location of the touch event, normalized
        (0...1) */
    float y; /**< the y-axis location of the touch event, normalized
        (0...1) */
    float pressure; /**< the quantity of pressure applied, normalized
        (0...1) */
} SDL_Finger;
```

Remarks

Each touch even is a collection of fingers that are simultaneously in contact with the touch device (so a "touch" can be a "multitouch," in reality), and this struct reports details of the specific fingers.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_GetTouchFinger

SDL_FlashOperation

Window flash operation.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef enum SDL_FlashOperation
{
    SDL_FLASH_CANCEL,                /**< Cancel any window flash state */
    SDL_FLASH_BRIEFLY,               /**< Flash the window briefly to get
        attention */
    SDL_FLASH_UNTIL_FOCUSED          /**< Flash the window until it gets
        focus */
} SDL_FlashOperation;
```

Version

This enum is available since SDL 3.0.0.

SDL_FlipMode

The flip mode.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
typedef enum SDL_FlipMode
{
    SDL_FLIP_NONE,        /**< Do not flip */
    SDL_FLIP_HORIZONTAL,  /**< flip horizontally */
    SDL_FLIP_VERTICAL     /**< flip vertically */
} SDL_FlipMode;
```

Version

This enum is available since SDL 3.0.0.

SDL_Folder

The type of the OS-provided default folder for a specific purpose.

Header File

Defined in SDL3/SDL_filesystem.h

Syntax

```
typedef enum SDL_Folder
{
    /** The folder which contains all of the current user's data,
        preferences,
      and documents. It usually contains most of the other folders. If a
      requested folder does not exist, the home folder can be considered
          a safe
      fallback to store a user's documents. */
    SDL_FOLDER_HOME,
    /** The folder of files that are displayed on the desktop. Note that
        the
      existence of a desktop folder does not guarantee that the system
          does
      show icons on its desktop; certain GNU/Linux distros with a
          graphical
      environment may not have desktop icons. */
    SDL_FOLDER_DESKTOP,
    /** User document files, possibly application-specific. This is a
        good
      place to save a user's projects. */
    SDL_FOLDER_DOCUMENTS,
    /** Standard folder for user files downloaded from the internet. */
    SDL_FOLDER_DOWNLOADS,
    /** Music files that can be played using a standard music player
        (mp3,
      ogg...). */
    SDL_FOLDER_MUSIC,
    /** Image files that can be displayed using a standard viewer (png,
      jpg...). */
    SDL_FOLDER_PICTURES,
    /** Files that are meant to be shared with other users on the same
      computer. */
    SDL_FOLDER_PUBLICSHARE,
    /** Save files for games. */
    SDL_FOLDER_SAVEDGAMES,
    /** Application screenshots. */
    SDL_FOLDER_SCREENSHOTS,
    /** Template files to be used when the user requests the desktop
        environment
```

```
    to create a new file in a certain folder, such as "New Text
        File.txt".
    Any file in the Templates folder can be used as a starting point
        for a
    new file. */
  SDL_FOLDER_TEMPLATES,
  /** Video files that can be played using a standard video player
      (mp4,
    webm...). */
  SDL_FOLDER_VIDEOS
} SDL_Folder;
```

Remarks

Note that the Trash folder isn't included here, because trashing files usually involves extra OS-specific functionality to remember the file's original location.

The folders supported per platform are:

— — Windows — WinRT/UWP —macOS/iOS — tvOS — Unix (XDG) — Haiku — Emscripten — — ——— — ——- — ——— ——— — —- — ———- — —— — ———- — — HOME — X — X — X — — X — X — X — — DESKTOP — X — X — X — — X — X — — — DOCUMENTS — X — X — X — — X — — — — DOWNLOADS — Vista+ — X — X — — X — — — — MUSIC — X — X — X — — X — — — — PICTURES — X — X — X — — X — — — — PUBLICSHARE — — — X — — X — — — — SAVEDGAMES — Vista+ — — — — — — — — SCREENSHOTS — Vista+ — X — — — — — — — TEMPLATES — X — X — X — — X — — — — VIDEOS — X — X — X* — — X — — —

Note that on macOS/iOS, the Videos folder is called "Movies".

Version

This enum is available since SDL 3.0.0.

See Also

- SDL_GetUserFolder

SDL_FPoint

The structure that defines a point (using floating point values).

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
typedef struct SDL_FPoint
{
    float x;
    float y;
} SDL_FPoint;
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_GetRectEnclosingPointsFloat
- SDL_PointInRectFloat

SDL_FRect

A rectangle, with the origin at the upper left (using floating point values).

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
typedef struct SDL_FRect
{
    float x;
    float y;
    float w;
    float h;
} SDL_FRect;
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_RectEmptyFloat
- SDL_RectsEqualFloat
- SDL_RectsEqualEpsilon
- SDL_HasRectIntersectionFloat
- SDL_GetRectIntersectionFloat
- SDL_GetRectAndLineIntersectionFloat
- SDL_GetRectUnionFloat
- SDL_GetRectEnclosingPointsFloat
- SDL_PointInRectFloat

SDL_Gamepad

The structure used to identify an SDL gamepad

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
typedef struct SDL_Gamepad SDL_Gamepad;
```

Version

This struct is available since SDL 3.0.0.

SDL_GamepadAxis

The list of axes available on a gamepad

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
typedef enum SDL_GamepadAxis
{
    SDL_GAMEPAD_AXIS_INVALID = -1,
    SDL_GAMEPAD_AXIS_LEFTX,
    SDL_GAMEPAD_AXIS_LEFTY,
    SDL_GAMEPAD_AXIS_RIGHTX,
    SDL_GAMEPAD_AXIS_RIGHTY,
    SDL_GAMEPAD_AXIS_LEFT_TRIGGER,
    SDL_GAMEPAD_AXIS_RIGHT_TRIGGER,
    SDL_GAMEPAD_AXIS_MAX
} SDL_GamepadAxis;
```

Remarks

Thumbstick axis values range from SDL_JOYSTICK_AXIS_MIN to SDL_JOYSTICK_AXIS_MAX, and are centered within 8000 of zero, though advanced UI will allow users to set or autodetect the dead zone, which varies between gamepads.

Trigger axis values range from 0 (released) to SDL_JOYSTICK_AXIS_MAX (fully pressed) when reported by SDL_GetGamepadAxis(). Note that this is not the same range that will be reported by the lower-level SDL_GetJoystickAxis().

Version

This enum is available since SDL 3.0.0.

SDL_GamepadAxisEvent

Gamepad axis motion event structure (event.gaxis.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_GamepadAxisEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_GAMEPAD_AXIS_MOTION */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which; /**< The joystick instance id */
    Uint8 axis;       /**< The gamepad axis (SDL_GamepadAxis) */
    Uint8 padding1;
    Uint8 padding2;
    Uint8 padding3;
    Sint16 value;     /**< The axis value (range: -32768 to 32767) */
    Uint16 padding4;
} SDL_GamepadAxisEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_GamepadBinding

A mapping between one joystick input to a gamepad control.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
typedef struct SDL_GamepadBinding
{
    SDL_GamepadBindingType input_type;
    union
    {
        int button;

        struct
        {
            int axis;
            int axis_min;
            int axis_max;
        } axis;

        struct
        {
            int hat;
            int hat_mask;
        } hat;

    } input;

    SDL_GamepadBindingType output_type;
    union
    {
        SDL_GamepadButton button;

        struct
        {
            SDL_GamepadAxis axis;
            int axis_min;
            int axis_max;
        } axis;

    } output;
} SDL_GamepadBinding;
```

Remarks

A gamepad has a collection of several bindings, to say, for example, when joystick button number 5 is pressed, that should be treated like the gamepad's "start" button.

SDL has these bindings built-in for many popular controllers, and can add more with a simple text string. Those strings are parsed into a collection of these structs to make it easier to operate on the data.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_GetGamepadBindings

SDL_GamepadBindingType

Types of gamepad control bindings.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
typedef enum SDL_GamepadBindingType
{
    SDL_GAMEPAD_BINDTYPE_NONE = 0,
    SDL_GAMEPAD_BINDTYPE_BUTTON,
    SDL_GAMEPAD_BINDTYPE_AXIS,
    SDL_GAMEPAD_BINDTYPE_HAT
} SDL_GamepadBindingType;
```

Remarks

A gamepad is a collection of bindings that map arbitrary joystick buttons, axes and hat switches to specific positions on a generic console-style gamepad. This enum is used as part of SDL_GamepadBinding to specify those mappings.

Version

This enum is available since SDL 3.0.0.

SDL_GamepadButton

The list of buttons available on a gamepad

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
typedef enum SDL_GamepadButton
{
    SDL_GAMEPAD_BUTTON_INVALID = -1,
    SDL_GAMEPAD_BUTTON_SOUTH,       /* Bottom face button (e.g. Xbox A
        button) */
    SDL_GAMEPAD_BUTTON_EAST,        /* Right face button (e.g. Xbox B
        button) */
    SDL_GAMEPAD_BUTTON_WEST,        /* Left face button (e.g. Xbox X
        button) */
    SDL_GAMEPAD_BUTTON_NORTH,       /* Top face button (e.g. Xbox Y
        button) */
    SDL_GAMEPAD_BUTTON_BACK,
    SDL_GAMEPAD_BUTTON_GUIDE,
    SDL_GAMEPAD_BUTTON_START,
    SDL_GAMEPAD_BUTTON_LEFT_STICK,
    SDL_GAMEPAD_BUTTON_RIGHT_STICK,
    SDL_GAMEPAD_BUTTON_LEFT_SHOULDER,
    SDL_GAMEPAD_BUTTON_RIGHT_SHOULDER,
    SDL_GAMEPAD_BUTTON_DPAD_UP,
    SDL_GAMEPAD_BUTTON_DPAD_DOWN,
    SDL_GAMEPAD_BUTTON_DPAD_LEFT,
    SDL_GAMEPAD_BUTTON_DPAD_RIGHT,
    SDL_GAMEPAD_BUTTON_MISC1,       /* Additional button (e.g. Xbox
        Series X share button, PS5 microphone button, Nintendo Switch
        Pro capture button, Amazon Luna microphone button, Google Stadia
        capture button) */
    SDL_GAMEPAD_BUTTON_RIGHT_PADDLE1, /* Upper or primary paddle, under
        your right hand (e.g. Xbox Elite paddle P1) */
    SDL_GAMEPAD_BUTTON_LEFT_PADDLE1, /* Upper or primary paddle, under
        your left hand (e.g. Xbox Elite paddle P3) */
    SDL_GAMEPAD_BUTTON_RIGHT_PADDLE2, /* Lower or secondary paddle,
        under your right hand (e.g. Xbox Elite paddle P2) */
    SDL_GAMEPAD_BUTTON_LEFT_PADDLE2, /* Lower or secondary paddle, under
        your left hand (e.g. Xbox Elite paddle P4) */
    SDL_GAMEPAD_BUTTON_TOUCHPAD,    /* PS4/PS5 touchpad button */
    SDL_GAMEPAD_BUTTON_MISC2,       /* Additional button */
    SDL_GAMEPAD_BUTTON_MISC3,       /* Additional button */
    SDL_GAMEPAD_BUTTON_MISC4,       /* Additional button */
```

```
    SDL_GAMEPAD_BUTTON_MISC5,        /* Additional button */
    SDL_GAMEPAD_BUTTON_MISC6,        /* Additional button */
    SDL_GAMEPAD_BUTTON_MAX
} SDL_GamepadButton;
```

Remarks

For controllers that use a diamond pattern for the face buttons, the south/east/west/north buttons below correspond to the locations in the diamond pattern. For Xbox controllers, this would be A/B/X/Y, for Nintendo Switch controllers, this would be B/A/Y/X, for PlayStation controllers this would be Cross/Circle/Square/Triangle.

For controllers that don't use a diamond pattern for the face buttons, the south/east/west/north buttons indicate the buttons labeled A, B, C, D, or 1, 2, 3, 4, or for controllers that aren't labeled, they are the primary, secondary, etc. buttons.

The activate action is often the south button and the cancel action is often the east button, but in some regions this is reversed, so your game should allow remapping actions based on user preferences.

You can query the labels for the face buttons using SDL_GetGamepadButtonLabel()

Version

This enum is available since SDL 3.0.0.

SDL_GamepadButtonEvent

Gamepad button event structure (event.gbutton.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_GamepadButtonEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_GAMEPAD_BUTTON_DOWN or
        ::SDL_EVENT_GAMEPAD_BUTTON_UP */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which; /**< The joystick instance id */
    Uint8 button;     /**< The gamepad button (SDL_GamepadButton) */
    Uint8 state;      /**< ::SDL_PRESSED or ::SDL_RELEASED */
    Uint8 padding1;
    Uint8 padding2;
} SDL_GamepadButtonEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_GamepadButtonLabel

The set of gamepad button labels

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
typedef enum SDL_GamepadButtonLabel
{
    SDL_GAMEPAD_BUTTON_LABEL_UNKNOWN,
    SDL_GAMEPAD_BUTTON_LABEL_A,
    SDL_GAMEPAD_BUTTON_LABEL_B,
    SDL_GAMEPAD_BUTTON_LABEL_X,
    SDL_GAMEPAD_BUTTON_LABEL_Y,
    SDL_GAMEPAD_BUTTON_LABEL_CROSS,
    SDL_GAMEPAD_BUTTON_LABEL_CIRCLE,
    SDL_GAMEPAD_BUTTON_LABEL_SQUARE,
    SDL_GAMEPAD_BUTTON_LABEL_TRIANGLE
} SDL_GamepadButtonLabel;
```

Remarks

This isn't a complete set, just the face buttons to make it easy to show button prompts.

For a complete set, you should look at the button and gamepad type and have a set of symbols that work well with your art style.

Version

This enum is available since SDL 3.0.0.

SDL_GamepadDeviceEvent

Gamepad device event structure (event.gdevice.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_GamepadDeviceEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_GAMEPAD_ADDED,
        ::SDL_EVENT_GAMEPAD_REMOVED, or ::SDL_EVENT_GAMEPAD_REMAPPED,
        ::SDL_EVENT_GAMEPAD_UPDATE_COMPLETE or
        ::SDL_EVENT_GAMEPAD_STEAM_HANDLE_UPDATED */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which;    /**< The joystick instance id */
} SDL_GamepadDeviceEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_GamepadSensorEvent

Gamepad sensor event structure (event.gsensor.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_GamepadSensorEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_GAMEPAD_SENSOR_UPDATE */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which; /**< The joystick instance id */
    Sint32 sensor;    /**< The type of the sensor, one of the values of
        ::SDL_SensorType */
    float data[3];    /**< Up to 3 values from the sensor, as defined in
        SDL_sensor.h */
    Uint64 sensor_timestamp; /**< The timestamp of the sensor reading in
        nanoseconds, not necessarily synchronized with the system clock
        */
} SDL_GamepadSensorEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_GamepadTouchpadEvent

Gamepad touchpad event structure (event.gtouchpad.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_GamepadTouchpadEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_GAMEPAD_TOUCHPAD_DOWN or
        ::SDL_EVENT_GAMEPAD_TOUCHPAD_MOTION or
        ::SDL_EVENT_GAMEPAD_TOUCHPAD_UP */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which; /**< The joystick instance id */
    Sint32 touchpad;  /**< The index of the touchpad */
    Sint32 finger;    /**< The index of the finger on the touchpad */
    float x;          /**< Normalized in the range 0...1 with 0 being on
        the left */
    float y;          /**< Normalized in the range 0...1 with 0 being at
        the top */
    float pressure;   /**< Normalized in the range 0...1 */
} SDL_GamepadTouchpadEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_GamepadType

Standard gamepad types.

Header File

Defined in SDL3/SDL_gamepad.h

Syntax

```
typedef enum SDL_GamepadType
{
    SDL_GAMEPAD_TYPE_UNKNOWN = 0,
    SDL_GAMEPAD_TYPE_STANDARD,
    SDL_GAMEPAD_TYPE_XBOX360,
    SDL_GAMEPAD_TYPE_XBOXONE,
    SDL_GAMEPAD_TYPE_PS3,
    SDL_GAMEPAD_TYPE_PS4,
    SDL_GAMEPAD_TYPE_PS5,
    SDL_GAMEPAD_TYPE_NINTENDO_SWITCH_PRO,
    SDL_GAMEPAD_TYPE_NINTENDO_SWITCH_JOYCON_LEFT,
    SDL_GAMEPAD_TYPE_NINTENDO_SWITCH_JOYCON_RIGHT,
    SDL_GAMEPAD_TYPE_NINTENDO_SWITCH_JOYCON_PAIR,
    SDL_GAMEPAD_TYPE_MAX
} SDL_GamepadType;
```

Remarks

This type does not necessarily map to first-party controllers from Microsoft/-Sony/Nintendo; in many cases, third-party controllers can report as these, either because they were designed for a specific console, or they simply most closely match that console's controllers (does it have A/B/X/Y buttons or X/O/Square/Triangle? Does it have a touchpad? etc).

SDL_GLattr

An enumeration of OpenGL configuration attributes.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef enum SDL_GLattr
{
    SDL_GL_RED_SIZE,
    SDL_GL_GREEN_SIZE,
    SDL_GL_BLUE_SIZE,
    SDL_GL_ALPHA_SIZE,
    SDL_GL_BUFFER_SIZE,
    SDL_GL_DOUBLEBUFFER,
    SDL_GL_DEPTH_SIZE,
    SDL_GL_STENCIL_SIZE,
    SDL_GL_ACCUM_RED_SIZE,
    SDL_GL_ACCUM_GREEN_SIZE,
    SDL_GL_ACCUM_BLUE_SIZE,
    SDL_GL_ACCUM_ALPHA_SIZE,
    SDL_GL_STEREO,
    SDL_GL_MULTISAMPLEBUFFERS,
    SDL_GL_MULTISAMPLESAMPLES,
    SDL_GL_ACCELERATED_VISUAL,
    SDL_GL_RETAINED_BACKING,
    SDL_GL_CONTEXT_MAJOR_VERSION,
    SDL_GL_CONTEXT_MINOR_VERSION,
    SDL_GL_CONTEXT_FLAGS,
    SDL_GL_CONTEXT_PROFILE_MASK,
    SDL_GL_SHARE_WITH_CURRENT_CONTEXT,
    SDL_GL_FRAMEBUFFER_SRGB_CAPABLE,
    SDL_GL_CONTEXT_RELEASE_BEHAVIOR,
    SDL_GL_CONTEXT_RESET_NOTIFICATION,
    SDL_GL_CONTEXT_NO_ERROR,
    SDL_GL_FLOATBUFFERS,
    SDL_GL_EGL_PLATFORM
} SDL_GLattr;
```

Remarks

While you can set most OpenGL attributes normally, the attributes listed above must be known before SDL creates the window that will be used with the

OpenGL context. These attributes are set and read with SDL_GL_SetAttribute() and SDL_GL_GetAttribute().

In some cases, these attributes are minimum requests; the GL does not promise to give you exactly what you asked for. It's possible to ask for a 16-bit depth buffer and get a 24-bit one instead, for example, or to ask for no stencil buffer and still have one available. Context creation should fail if the GL can't provide your requested attributes at a minimum, but you should check to see exactly what you got.

Version

This enum is available since SDL 3.0.0.

SDL_GLContext

An opaque handle to an OpenGL context.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef void *SDL_GLContext;
```

Version

This datatype is available since SDL 3.0.0.

SDL_GLcontextFlag

Possible values to be set for the SDL_GL_CONTEXT_FLAGS attribute.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef enum SDL_GLcontextFlag
{
    SDL_GL_CONTEXT_DEBUG_FLAG          = 0x0001,
    SDL_GL_CONTEXT_FORWARD_COMPATIBLE_FLAG = 0x0002,
    SDL_GL_CONTEXT_ROBUST_ACCESS_FLAG  = 0x0004,
    SDL_GL_CONTEXT_RESET_ISOLATION_FLAG = 0x0008
} SDL_GLcontextFlag;
```

Version

This enum is available since SDL 3.0.0.

SDL_GLcontextReleaseFlag

Possible values to be set for the SDL_GL_CONTEXT_RELEASE_BEHAVIOR attribute.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef enum SDL_GLcontextReleaseFlag
{
    SDL_GL_CONTEXT_RELEASE_BEHAVIOR_NONE = 0x0000,
    SDL_GL_CONTEXT_RELEASE_BEHAVIOR_FLUSH = 0x0001
} SDL_GLcontextReleaseFlag;
```

Version

This enum is available since SDL 3.0.0.

SDL_GLContextResetNotification

Possible values to be set SDL_GL_CONTEXT_RESET_NOTIFICATION attribute.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef enum SDL_GLContextResetNotification
{
    SDL_GL_CONTEXT_RESET_NO_NOTIFICATION = 0x0000,
    SDL_GL_CONTEXT_RESET_LOSE_CONTEXT = 0x0001
} SDL_GLContextResetNotification;
```

Version

This enum is available since SDL 3.0.0.

SDL_GLprofile

Possible values to be set for the SDL_GL_CONTEXT_PROFILE_MASK attribute.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef enum SDL_GLprofile
{
    SDL_GL_CONTEXT_PROFILE_CORE       = 0x0001,
    SDL_GL_CONTEXT_PROFILE_COMPATIBILITY = 0x0002,
    SDL_GL_CONTEXT_PROFILE_ES         = 0x0004 /**<
        GLX_CONTEXT_ES2_PROFILE_BIT_EXT */
} SDL_GLprofile;
```

Version

This enum is available since SDL 3.0.0.

SDL_GUID

An SDL_GUID is a 128-bit identifier for an input device that identifies that device across runs of SDL programs on the same platform.

Header File

Defined in SDL3/SDL_guid.h

Syntax

```
typedef struct SDL_GUID {
    Uint8 data[16];
} SDL_GUID;
```

Remarks

If the device is detached and then re-attached to a different port, or if the base system is rebooted, the device should still report the same GUID.

GUIDs are as precise as possible but are not guaranteed to distinguish physically distinct but equivalent devices. For example, two game controllers from the same vendor with the same product ID and revision may have the same GUID.

GUIDs may be platform-dependent (i.e., the same device may report different GUIDs on different operating systems).

Version

This struct is available since SDL 3.0.0.

SDL_HapticCondition

A structure containing a template for a Condition effect.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
typedef struct SDL_HapticCondition
{
    /* Header */
    Uint16 type;          /**< SDL_HAPTIC_SPRING, SDL_HAPTIC_DAMPER,
                               SDL_HAPTIC_INERTIA or SDL_HAPTIC_FRICTION
                                   */
    SDL_HapticDirection direction; /**< Direction of the effect - Not
        used ATM. */

    /* Replay */
    Uint32 length;        /**< Duration of the effect. */
    Uint16 delay;         /**< Delay before starting the effect. */

    /* Trigger */
    Uint16 button;        /**< Button that triggers the effect. */
    Uint16 interval;      /**< How soon it can be triggered again after
        button. */

    /* Condition */
    Uint16 right_sat[3];  /**< Level when joystick is to the positive
        side; max 0xFFFF. */
    Uint16 left_sat[3];   /**< Level when joystick is to the negative
        side; max 0xFFFF. */
    Sint16 right_coeff[3]; /**< How fast to increase the force towards
        the positive side. */
    Sint16 left_coeff[3]; /**< How fast to increase the force towards
        the negative side. */
    Uint16 deadband[3];   /**< Size of the dead zone; max 0xFFFF: whole
        axis-range when 0-centered. */
    Sint16 center[3];     /**< Position of the dead zone. */
} SDL_HapticCondition;
```

Remarks

The struct handles the following effects:

- SDL_HAPTIC_SPRING: Effect based on axes position.

- SDL_HAPTIC_DAMPER: Effect based on axes velocity.
- SDL_HAPTIC_INERTIA: Effect based on axes acceleration.
- SDL_HAPTIC_FRICTION: Effect based on axes movement.

Direction is handled by condition internals instead of a direction member. The condition effect specific members have three parameters. The first refers to the X axis, the second refers to the Y axis and the third refers to the Z axis. The right terms refer to the positive side of the axis and the left terms refer to the negative side of the axis. Please refer to the SDL_HapticDirection diagram for which side is positive and which is negative.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_HapticDirection
- SDL_HAPTIC_SPRING
- SDL_HAPTIC_DAMPER
- SDL_HAPTIC_INERTIA
- SDL_HAPTIC_FRICTION
- SDL_HapticEffect

SDL_HapticConstant

A structure containing a template for a Constant effect.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
typedef struct SDL_HapticConstant
{
    /* Header */
    Uint16 type;          /**< SDL_HAPTIC_CONSTANT */
    SDL_HapticDirection direction; /**< Direction of the effect. */

    /* Replay */
    Uint32 length;        /**< Duration of the effect. */
    Uint16 delay;         /**< Delay before starting the effect. */

    /* Trigger */
    Uint16 button;        /**< Button that triggers the effect. */
    Uint16 interval;      /**< How soon it can be triggered again after
        button. */

    /* Constant */
    Sint16 level;         /**< Strength of the constant effect. */

    /* Envelope */
    Uint16 attack_length; /**< Duration of the attack. */
    Uint16 attack_level;  /**< Level at the start of the attack. */
    Uint16 fade_length;   /**< Duration of the fade. */
    Uint16 fade_level;    /**< Level at the end of the fade. */
} SDL_HapticConstant;
```

Remarks

This struct is exclusively for the SDL_HAPTIC_CONSTANT effect.

A constant effect applies a constant force in the specified direction to the joystick.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_HAPTIC_CONSTANT
- SDL_HapticEffect

SDL_HapticCustom

A structure containing a template for the SDL_HAPTIC_CUSTOM effect.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
typedef struct SDL_HapticCustom
{
    /* Header */
    Uint16 type;          /**< SDL_HAPTIC_CUSTOM */
    SDL_HapticDirection direction; /**< Direction of the effect. */

    /* Replay */
    Uint32 length;        /**< Duration of the effect. */
    Uint16 delay;         /**< Delay before starting the effect. */

    /* Trigger */
    Uint16 button;        /**< Button that triggers the effect. */
    Uint16 interval;      /**< How soon it can be triggered again after
        button. */

    /* Custom */
    Uint8 channels;       /**< Axes to use, minimum of one. */
    Uint16 period;        /**< Sample periods. */
    Uint16 samples;       /**< Amount of samples. */
    Uint16 *data;         /**< Should contain channels*samples items. */

    /* Envelope */
    Uint16 attack_length; /**< Duration of the attack. */
    Uint16 attack_level;  /**< Level at the start of the attack. */
    Uint16 fade_length;   /**< Duration of the fade. */
    Uint16 fade_level;    /**< Level at the end of the fade. */
} SDL_HapticCustom;
```

Remarks

This struct is exclusively for the SDL_HAPTIC_CUSTOM effect.

A custom force feedback effect is much like a periodic effect, where the application can define its exact shape. You will have to allocate the data yourself. Data should consist of channels * samples Uint16 samples.

If channels is one, the effect is rotated using the defined direction. Otherwise it uses the samples in data for the different axes.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_HAPTIC_CUSTOM
- SDL_HapticEffect

SDL_HapticDirection

Structure that represents a haptic direction.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
typedef struct SDL_HapticDirection
{
    Uint8 type;       /**< The type of encoding. */
    Sint32 dir[3];    /**< The encoded direction. */
} SDL_HapticDirection;
```

Remarks

This is the direction where the force comes from, instead of the direction in which the force is exerted.

Directions can be specified by:

- SDL_HAPTIC_POLAR : Specified by polar coordinates.
- SDL_HAPTIC_CARTESIAN : Specified by cartesian coordinates.
- SDL_HAPTIC_SPHERICAL : Specified by spherical coordinates.

Cardinal directions of the haptic device are relative to the positioning of the device. North is considered to be away from the user.

The following diagram represents the cardinal directions:

```
               .--.
               |__| .-------.
               |=.| |.-----.|
               |--| ||     ||
               |  | |'-----'|
               |__|~')_____('
                 [ COMPUTER ]


                   North (0,-1)
                       ^
                       |
                       |
 (-1,0)  West <----[ HAPTIC ]----> East (1,0)
```

```
                    |
                    |
                    v
                 South (0,1)


                 [ USER ]
                   \|||/
                   (o o)
           ---ooO-(_)-Ooo---
```

If type is SDL_HAPTIC_POLAR, direction is encoded by hundredths of a degree starting north and turning clockwise. SDL_HAPTIC_POLAR only uses the first `dir` parameter. The cardinal directions would be:

- North: 0 (0 degrees)
- East: 9000 (90 degrees)
- South: 18000 (180 degrees)
- West: 27000 (270 degrees)

If type is SDL_HAPTIC_CARTESIAN, direction is encoded by three positions (X axis, Y axis and Z axis (with 3 axes)). SDL_HAPTIC_CARTESIAN uses the first three `dir` parameters. The cardinal directions would be:

- North: 0,-1, 0
- East: 1, 0, 0
- South: 0, 1, 0
- West: -1, 0, 0

The Z axis represents the height of the effect if supported, otherwise it's unused. In cartesian encoding (1, 2) would be the same as (2, 4), you can use any multiple you want, only the direction matters.

If type is SDL_HAPTIC_SPHERICAL, direction is encoded by two rotations. The first two `dir` parameters are used. The `dir` parameters are as follows (all values are in hundredths of degrees):

- Degrees from (1, 0) rotated towards (0, 1).
- Degrees towards (0, 0, 1) (device needs at least 3 axes).

Example of force coming from the south with all encodings (force coming from the south means the user will have to pull the stick to counteract):

```
SDL_HapticDirection direction;

// Cartesian directions
direction.type = SDL_HAPTIC_CARTESIAN; // Using cartesian direction
    encoding.
direction.dir[0] = 0; // X position
direction.dir[1] = 1; // Y position
// Assuming the device has 2 axes, we don't need to specify third
    parameter.

// Polar directions
direction.type = SDL_HAPTIC_POLAR; // We'll be using polar direction
    encoding.
direction.dir[0] = 18000; // Polar only uses first parameter

// Spherical coordinates
direction.type = SDL_HAPTIC_SPHERICAL; // Spherical encoding
direction.dir[0] = 9000; // Since we only have two axes we don't need
    more parameters.
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_HAPTIC_POLAR
- SDL_HAPTIC_CARTESIAN
- SDL_HAPTIC_SPHERICAL
- SDL_HAPTIC_STEERING_AXIS
- SDL_HapticEffect
- SDL_GetNumHapticAxes

SDL_HapticEffect

The generic template for any haptic effect.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
typedef union SDL_HapticEffect
{
    /* Common for all force feedback effects */
    Uint16 type;                /**< Effect type. */
    SDL_HapticConstant constant; /**< Constant effect. */
    SDL_HapticPeriodic periodic; /**< Periodic effect. */
    SDL_HapticCondition condition; /**< Condition effect. */
    SDL_HapticRamp ramp;        /**< Ramp effect. */
    SDL_HapticLeftRight leftright; /**< Left/Right effect. */
    SDL_HapticCustom custom;    /**< Custom effect. */
} SDL_HapticEffect;
```

Remarks

All values max at 32767 (0x7FFF). Signed values also can be negative. Time values unless specified otherwise are in milliseconds.

You can also pass SDL_HAPTIC_INFINITY to length instead of a 0-32767 value. Neither delay, interval, attack_length nor fade_length support SDL_HAPTIC_INFINITY. Fade will also not be used since effect never ends.

Additionally, the SDL_HAPTIC_RAMP effect does not support a duration of SDL_HAPTIC_INFINITY.

Button triggers may not be supported on all devices, it is advised to not use them if possible. Buttons start at index 1 instead of index 0 like the joystick.

If both attack_length and fade_level are 0, the envelope is not used, otherwise both values are used.

Common parts:

```
 // Replay - All effects have this
 Uint32 length;      // Duration of effect (ms).
 Uint16 delay;       // Delay before starting effect.

 // Trigger - All effects have this
 Uint16 button;      // Button that triggers effect.
 Uint16 interval;    // How soon before effect can be triggered again.

 // Envelope - All effects except condition effects have this
```

```
Uint16 attack_length; // Duration of the attack (ms).
Uint16 attack_level; // Level at the start of the attack.
Uint16 fade_length; // Duration of the fade out (ms).
Uint16 fade_level;  // Level at the end of the fade.
```

Here we have an example of a constant effect evolution in time:

```
Strength
^
|
|    effect level -->  _________________
|                     /                 \
|                    /                   \
|                   /                     \
|                  /                       \
| attack_level --> |                        \
|                  |                         |  <---  fade_level
|
+-------------------------------------------------> Time
                   [--]                 [---]
                   attack_length        fade_length

[------------------][-----------------------]
delay               length
```

Note either the attack_level or the fade_level may be above the actual effect level.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_HapticConstant
- SDL_HapticPeriodic
- SDL_HapticCondition
- SDL_HapticRamp
- SDL_HapticLeftRight
- SDL_HapticCustom

SDL_HapticID

This is a unique ID for a haptic device for the time it is connected to the system, and is never reused for the lifetime of the application.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
typedef Uint32 SDL_HapticID;
```

Remarks

If the haptic device is disconnected and reconnected, it will get a new ID.

The ID value starts at 1 and increments from there. The value 0 is an invalid ID.

Version

This datatype is available since SDL 3.0.0.

SDL_HapticLeftRight

A structure containing a template for a Left/Right effect.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
typedef struct SDL_HapticLeftRight
{
    /* Header */
    Uint16 type;          /**< SDL_HAPTIC_LEFTRIGHT */

    /* Replay */
    Uint32 length;        /**< Duration of the effect in milliseconds. */

    /* Rumble */
    Uint16 large_magnitude; /**< Control of the large controller motor.
        */
    Uint16 small_magnitude; /**< Control of the small controller motor.
        */
} SDL_HapticLeftRight;
```

Remarks

This struct is exclusively for the SDL_HAPTIC_LEFTRIGHT effect.

The Left/Right effect is used to explicitly control the large and small motors, commonly found in modern game controllers. The small (right) motor is high frequency, and the large (left) motor is low frequency.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_HAPTIC_LEFTRIGHT
- SDL_HapticEffect

SDL_HapticPeriodic

A structure containing a template for a Periodic effect.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
typedef struct SDL_HapticPeriodic
{
    /* Header */
    Uint16 type;      /**< SDL_HAPTIC_SINE, SDL_HAPTIC_SQUARE
                          SDL_HAPTIC_TRIANGLE, SDL_HAPTIC_SAWTOOTHUP or
                          SDL_HAPTIC_SAWTOOTHDOWN */
    SDL_HapticDirection direction; /**< Direction of the effect. */

    /* Replay */
    Uint32 length;    /**< Duration of the effect. */
    Uint16 delay;     /**< Delay before starting the effect. */

    /* Trigger */
    Uint16 button;    /**< Button that triggers the effect. */
    Uint16 interval;  /**< How soon it can be triggered again after
        button. */

    /* Periodic */
    Uint16 period;    /**< Period of the wave. */
    Sint16 magnitude; /**< Peak value; if negative, equivalent to 180
        degrees extra phase shift. */
    Sint16 offset;    /**< Mean value of the wave. */
    Uint16 phase;     /**< Positive phase shift given by hundredth of a
        degree. */

    /* Envelope */
    Uint16 attack_length; /**< Duration of the attack. */
    Uint16 attack_level;  /**< Level at the start of the attack. */
    Uint16 fade_length; /**< Duration of the fade. */
    Uint16 fade_level; /**< Level at the end of the fade. */
} SDL_HapticPeriodic;
```

Remarks

The struct handles the following effects:

- SDL_HAPTIC_SINE

- SDL_HAPTIC_SQUARE
- SDL_HAPTIC_TRIANGLE
- SDL_HAPTIC_SAWTOOTHUP
- SDL_HAPTIC_SAWTOOTHDOWN

A periodic effect consists in a wave-shaped effect that repeats itself over time. The type determines the shape of the wave and the parameters determine the dimensions of the wave.

Phase is given by hundredth of a degree meaning that giving the phase a value of 9000 will displace it 25

- 0: No phase displacement.
- 9000: Displaced 25
- 18000: Displaced 50
- 27000: Displaced 75
- 36000: Displaced 100

Examples:

```
SDL_HAPTIC_SINE
  __      __      __      __
 /  \    /  \    /  \    /
/    \__/    \__/    \__/

SDL_HAPTIC_SQUARE
 __    __    __    __    __
|  |  |  |  |  |  |  |  |  |
|  |__|  |__|  |__|  |__|  |

SDL_HAPTIC_TRIANGLE
  /\    /\    /\    /\    /\
 /  \  /  \  /  \  /  \  /
/    \/    \/    \/    \/

SDL_HAPTIC_SAWTOOTHUP
  /|  /|  /|  /|  /|  /|  /|
 / | / | / | / | / | / | / |
/  |/  |/  |/  |/  |/  |/  |

SDL_HAPTIC_SAWTOOTHDOWN
\  |\  |\  |\  |\  |\  |\  |
 \ | \ | \ | \ | \ | \ | \ |
  \|  \|  \|  \|  \|  \|  \|
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_HAPTIC_SINE
- SDL_HAPTIC_SQUARE
- SDL_HAPTIC_TRIANGLE
- SDL_HAPTIC_SAWTOOTHUP
- SDL_HAPTIC_SAWTOOTHDOWN
- SDL_HapticEffect

SDL_HapticRamp

A structure containing a template for a Ramp effect.

Header File

Defined in SDL3/SDL_haptic.h

Syntax

```
typedef struct SDL_HapticRamp
{
    /* Header */
    Uint16 type;          /**< SDL_HAPTIC_RAMP */
    SDL_HapticDirection direction; /**< Direction of the effect. */

    /* Replay */
    Uint32 length;        /**< Duration of the effect. */
    Uint16 delay;         /**< Delay before starting the effect. */

    /* Trigger */
    Uint16 button;        /**< Button that triggers the effect. */
    Uint16 interval;      /**< How soon it can be triggered again after
        button. */

    /* Ramp */
    Sint16 start;         /**< Beginning strength level. */
    Sint16 end;           /**< Ending strength level. */

    /* Envelope */
    Uint16 attack_length; /**< Duration of the attack. */
    Uint16 attack_level;  /**< Level at the start of the attack. */
    Uint16 fade_length;   /**< Duration of the fade. */
    Uint16 fade_level;    /**< Level at the end of the fade. */
} SDL_HapticRamp;
```

Remarks

This struct is exclusively for the SDL_HAPTIC_RAMP effect.

The ramp effect starts at start strength and ends at end strength. It augments in linear fashion. If you use attack and fade with a ramp the effects get added to the ramp effect making the effect become quadratic instead of linear.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_HAPTIC_RAMP
- SDL_HapticEffect

SDL_hid_bus_type

HID underlying bus types.

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
typedef enum SDL_hid_bus_type {
    /** Unknown bus type */
    SDL_HID_API_BUS_UNKNOWN = 0x00,

    /** USB bus
       Specifications:
       https://usb.org/hid */
    SDL_HID_API_BUS_USB = 0x01,

    /** Bluetooth or Bluetooth LE bus
       Specifications:
       https://www.bluetooth.com/specifications/specs/human-interface-device-profile-1-1-1/
       https://www.bluetooth.com/specifications/specs/hid-service-1-0/
       https://www.bluetooth.com/specifications/specs/hid-over-gatt-profile-1-0/
           */
    SDL_HID_API_BUS_BLUETOOTH = 0x02,

    /** I2C bus
       Specifications:
       https://docs.microsoft.com/previous-versions/windows/hardware/design/dn642101(v=vs.85)
           */
    SDL_HID_API_BUS_I2C = 0x03,

    /** SPI bus
       Specifications:
       https://www.microsoft.com/download/details.aspx?id=103325 */
    SDL_HID_API_BUS_SPI = 0x04

} SDL_hid_bus_type;
```

Version

This enum is available since SDL 3.0.0.

SDL_hid_device

A handle representing an open HID device

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
struct SDL_hid_device;
```

Version

This struct is available since SDL 3.0.0.

SDL_hid_device_info

Information about a connected HID device

Header File

Defined in SDL3/SDL_hidapi.h

Syntax

```
typedef struct SDL_hid_device_info
{
    /** Platform-specific device path */
    char *path;
    /** Device Vendor ID */
    unsigned short vendor_id;
    /** Device Product ID */
    unsigned short product_id;
    /** Serial Number */
    wchar_t *serial_number;
    /** Device Release Number in binary-coded decimal,
        also known as Device Version Number */
    unsigned short release_number;
    /** Manufacturer String */
    wchar_t *manufacturer_string;
    /** Product string */
    wchar_t *product_string;
    /** Usage Page for this Device/Interface
        (Windows/Mac/hidraw only) */
    unsigned short usage_page;
    /** Usage for this Device/Interface
        (Windows/Mac/hidraw only) */
    unsigned short usage;
    /** The USB interface which this logical device
        represents.

        Valid only if the device is a USB HID device.
        Set to -1 in all other cases.
    */
    int interface_number;

    /** Additional information about the USB interface.
        Valid on libusb and Android implementations. */
    int interface_class;
    int interface_subclass;
    int interface_protocol;

    /** Underlying bus type */
```

```
    SDL_hid_bus_type bus_type;

    /** Pointer to the next device */
    struct SDL_hid_device_info *next;

} SDL_hid_device_info;
```

Version

This struct is available since SDL 3.0.0.

SDL_HintCallback

Type definition of the hint callback function.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
typedef void (SDLCALL *SDL_HintCallback)(void *userdata, const char
    *name, const char *oldValue, const char *newValue);
```

Function Parameters

userdata	what was passed as `userdata` to SDL_AddHintCallback()
name	what was passed as `name` to SDL_AddHintCallback()
oldValue	the previous hint value
newValue	the new value hint is to be set to

Version

This datatype is available since SDL 3.0.0.

SDL_HintPriority

An enumeration of hint priorities.

Header File

Defined in SDL3/SDL_hints.h

Syntax

```
typedef enum SDL_HintPriority
{
    SDL_HINT_DEFAULT,
    SDL_HINT_NORMAL,
    SDL_HINT_OVERRIDE
} SDL_HintPriority;
```

Version

This enum is available since SDL 3.0.0.

SDL_HitTest

Callback used for hit-testing.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef SDL_HitTestResult (SDLCALL *SDL_HitTest)(SDL_Window *win, const
    SDL_Point *area, void *data);
```

Function Parameters

win	the SDL_Window where hit-testing was set on
area	an SDL_Point which should be hit-tested
data	what was passed as `callback_data` to SDL_SetWindowHitTest()

Return Value

Returns an SDL_HitTestResult value.

See Also

- SDL_SetWindowHitTest

SDL_HitTestResult

Possible return values from the SDL_HitTest callback.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef enum SDL_HitTestResult
{
    SDL_HITTEST_NORMAL,          /**< Region is normal. No special
        properties. */
    SDL_HITTEST_DRAGGABLE,       /**< Region can drag entire window. */
    SDL_HITTEST_RESIZE_TOPLEFT,  /**< Region is the resizable top-left
        corner border. */
    SDL_HITTEST_RESIZE_TOP,      /**< Region is the resizable top
        border. */
    SDL_HITTEST_RESIZE_TOPRIGHT, /**< Region is the resizable top-right
        corner border. */
    SDL_HITTEST_RESIZE_RIGHT,    /**< Region is the resizable right
        border. */
    SDL_HITTEST_RESIZE_BOTTOMRIGHT, /**< Region is the resizable
        bottom-right corner border. */
    SDL_HITTEST_RESIZE_BOTTOM,   /**< Region is the resizable bottom
        border. */
    SDL_HITTEST_RESIZE_BOTTOMLEFT, /**< Region is the resizable
        bottom-left corner border. */
    SDL_HITTEST_RESIZE_LEFT      /**< Region is the resizable left
        border. */
} SDL_HitTestResult;
```

Version

This enum is available since SDL 3.0.0.

See Also

- SDL_HitTest

SDL_InitFlags

Initialization flags for SDL_Init and/or SDL_InitSubSystem

Header File

Defined in SDL3/SDL_init.h

Syntax

```
typedef enum SDL_InitFlags
{
    SDL_INIT_TIMER      = 0x00000001,
    SDL_INIT_AUDIO      = 0x00000010, /**< ‘SDL_INIT_AUDIO‘ implies
        ‘SDL_INIT_EVENTS‘ */
    SDL_INIT_VIDEO      = 0x00000020, /**< ‘SDL_INIT_VIDEO‘ implies
        ‘SDL_INIT_EVENTS‘ */
    SDL_INIT_JOYSTICK   = 0x00000200, /**< ‘SDL_INIT_JOYSTICK‘ implies
        ‘SDL_INIT_EVENTS‘, should be initialized on the same thread as
        SDL_INIT_VIDEO on Windows if you don’t set
        SDL_HINT_JOYSTICK_THREAD */
    SDL_INIT_HAPTIC     = 0x00001000,
    SDL_INIT_GAMEPAD    = 0x00002000, /**< ‘SDL_INIT_GAMEPAD‘ implies
        ‘SDL_INIT_JOYSTICK‘ */
    SDL_INIT_EVENTS     = 0x00004000,
    SDL_INIT_SENSOR     = 0x00008000, /**< ‘SDL_INIT_SENSOR‘ implies
        ‘SDL_INIT_EVENTS‘ */
    SDL_INIT_CAMERA     = 0x00010000  /**< ‘SDL_INIT_CAMERA‘ implies
        ‘SDL_INIT_EVENTS‘ */
} SDL_InitFlags;
```

Remarks

These are the flags which may be passed to SDL_Init(). You should specify the subsystems which you will be using in your application.

Version

This enum is available since SDL 3.0.0.

See Also

- SDL_Init
- SDL_Quit
- SDL_InitSubSystem

- SDL_QuitSubSystem
- SDL_WasInit

SDL_IOStream

The read/write operation structure.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
typedef struct SDL_IOStream SDL_IOStream;
```

Remarks

This operates as an opaque handle. There are several APIs to create various types of I/O streams, or an app can supply an SDL_IOStreamInterface to SDL_OpenIO() to provide their own stream implementation behind this struct's abstract interface.

Version

This struct is available since SDL 3.0.0.

SDL_IOStreamInterface

The function pointers that drive an SDL_IOStream.

Header File

Defined in SDL3/SDL_iostream.h

Syntax

```
typedef struct SDL_IOStreamInterface
{
    /**
     * Return the number of bytes in this SDL_IOStream
     *
     * \return the total size of the data stream, or -1 on error.
     */
    Sint64 (SDLCALL *size)(void *userdata);

    /**
     * Seek to `offset` relative to `whence`, one of stdio's whence
       values:
     * SDL_IO_SEEK_SET, SDL_IO_SEEK_CUR, SDL_IO_SEEK_END
     *
     * \return the final offset in the data stream, or -1 on error.
     */
    Sint64 (SDLCALL *seek)(void *userdata, Sint64 offset, int whence);

    /**
     * Read up to `size` bytes from the data stream to the area pointed
     * at by `ptr`.
     *
     * On an incomplete read, you should set `*status` to a value from
       the
     * SDL_IOStatus enum. You do not have to explicitly set this on
     * a complete, successful read.
     *
     * \return the number of bytes read
     */
    size_t (SDLCALL *read)(void *userdata, void *ptr, size_t size,
        SDL_IOStatus *status);

    /**
     * Write exactly `size` bytes from the area pointed at by `ptr`
     * to data stream.
     *
     * On an incomplete write, you should set `*status` to a value from
       the
```

```
     *  SDL_IOStatus enum. You do not have to explicitly set this on
     *  a complete, successful write.
     *
     *  \return the number of bytes written
     */
    size_t (SDLCALL *write)(void *userdata, const void *ptr, size_t
        size, SDL_IOStatus *status);

    /**
     *  Close and free any allocated resources.
     *
     *  The SDL_IOStream is still destroyed even if this fails, so clean
         up anything
     *  even if flushing to disk returns an error.
     *
     *  \return 0 if successful or -1 on write error when flushing data.
     */
    int (SDLCALL *close)(void *userdata);
} SDL_IOStreamInterface;
```

Remarks

Applications can provide this struct to SDL_OpenIO() to create their own implementation of SDL_IOStream. This is not necessarily required, as SDL already offers several common types of I/O streams, via functions like SDL_IOFromFile() and SDL_IOFromMem().

Version

This struct is available since SDL 3.0.0.

SDL_JoyAxisEvent

Joystick axis motion event structure (event.jaxis.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_JoyAxisEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_JOYSTICK_AXIS_MOTION */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which; /**< The joystick instance id */
    Uint8 axis;       /**< The joystick axis index */
    Uint8 padding1;
    Uint8 padding2;
    Uint8 padding3;
    Sint16 value;     /**< The axis value (range: -32768 to 32767) */
    Uint16 padding4;
} SDL_JoyAxisEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_JoyBallEvent

Joystick trackball motion event structure (event.jball.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_JoyBallEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_JOYSTICK_BALL_MOTION */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which; /**< The joystick instance id */
    Uint8 ball;       /**< The joystick trackball index */
    Uint8 padding1;
    Uint8 padding2;
    Uint8 padding3;
    Sint16 xrel;      /**< The relative motion in the X direction */
    Sint16 yrel;      /**< The relative motion in the Y direction */
} SDL_JoyBallEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_JoyBatteryEvent

Joysick battery level change event structure (event.jbattery.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_JoyBatteryEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_JOYSTICK_BATTERY_UPDATED */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which; /**< The joystick instance id */
    SDL_PowerState state; /**< The joystick battery state */
    int percent;        /**< The joystick battery percent charge
        remaining */
} SDL_JoyBatteryEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_JoyButtonEvent

Joystick button event structure (event.jbutton.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_JoyButtonEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_JOYSTICK_BUTTON_DOWN or
        ::SDL_EVENT_JOYSTICK_BUTTON_UP */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which; /**< The joystick instance id */
    Uint8 button;     /**< The joystick button index */
    Uint8 state;      /**< ::SDL_PRESSED or ::SDL_RELEASED */
    Uint8 padding1;
    Uint8 padding2;
} SDL_JoyButtonEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_JoyDeviceEvent

Joystick device event structure (event.jdevice.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_JoyDeviceEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_JOYSTICK_ADDED or
        ::SDL_EVENT_JOYSTICK_REMOVED or
        ::SDL_EVENT_JOYSTICK_UPDATE_COMPLETE */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which;    /**< The joystick instance id */
} SDL_JoyDeviceEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_JoyHatEvent

Joystick hat position change event structure (event.jhat.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_JoyHatEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_JOYSTICK_HAT_MOTION */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_JoystickID which; /**< The joystick instance id */
    Uint8 hat;        /**< The joystick hat index */
    Uint8 value;      /**< The hat position value.
                       *   \sa ::SDL_HAT_LEFTUP ::SDL_HAT_UP
                           ::SDL_HAT_RIGHTUP
                       *   \sa ::SDL_HAT_LEFT ::SDL_HAT_CENTERED
                           ::SDL_HAT_RIGHT
                       *   \sa ::SDL_HAT_LEFTDOWN ::SDL_HAT_DOWN
                           ::SDL_HAT_RIGHTDOWN
                       *
                       *   Note that zero means the POV is centered.
                       */
    Uint8 padding1;
    Uint8 padding2;
} SDL_JoyHatEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_Joystick

The joystick structure used to identify an SDL joystick.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
typedef struct SDL_Joystick SDL_Joystick;
```

Remarks

This is opaque data.

Version

This struct is available since SDL 3.0.0.

SDL_JoystickConnectionState

Possible connection states for a joystick device.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
typedef enum SDL_JoystickConnectionState
{
    SDL_JOYSTICK_CONNECTION_INVALID = -1,
    SDL_JOYSTICK_CONNECTION_UNKNOWN,
    SDL_JOYSTICK_CONNECTION_WIRED,
    SDL_JOYSTICK_CONNECTION_WIRELESS
} SDL_JoystickConnectionState;
```

Remarks

This is used by SDL_GetJoystickConnectionState to report how a device is connected to the system.

Version

This enum is available since SDL 3.0.0.

SDL_JoystickGUID

A structure that encodes the stable unique id for a joystick device.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
typedef SDL_GUID SDL_JoystickGUID;
```

Remarks

This is just a standard SDL_GUID by a different name.

Version

This datatype is available since SDL 3.0.0.

SDL_JoystickID

This is a unique ID for a joystick for the time it is connected to the system, and is never reused for the lifetime of the application.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
typedef Uint32 SDL_JoystickID;
```

Remarks

If the joystick is disconnected and reconnected, it will get a new ID.

The ID value starts at 1 and increments from there. The value 0 is an invalid ID.

Version

This datatype is available since SDL 3.0.0.

SDL_JoystickType

An enum of some common joystick types.

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
typedef enum SDL_JoystickType
{
    SDL_JOYSTICK_TYPE_UNKNOWN,
    SDL_JOYSTICK_TYPE_GAMEPAD,
    SDL_JOYSTICK_TYPE_WHEEL,
    SDL_JOYSTICK_TYPE_ARCADE_STICK,
    SDL_JOYSTICK_TYPE_FLIGHT_STICK,
    SDL_JOYSTICK_TYPE_DANCE_PAD,
    SDL_JOYSTICK_TYPE_GUITAR,
    SDL_JOYSTICK_TYPE_DRUM_KIT,
    SDL_JOYSTICK_TYPE_ARCADE_PAD,
    SDL_JOYSTICK_TYPE_THROTTLE
} SDL_JoystickType;
```

Remarks

In some cases, SDL can identify a low-level joystick as being a certain type of device, and will report it through SDL_GetJoystickType (or SDL_GetJoystickInstanceType).

This is by no means a complete list of everything that can be plugged into a computer.

Version

This enum is available since SDL 3.0.0.

SDL_KeyboardDeviceEvent

Keyboard device event structure (event.kdevice.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_KeyboardDeviceEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_KEYBOARD_ADDED or
        ::SDL_EVENT_KEYBOARD_REMOVED */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_KeyboardID which; /**< The keyboard instance id */
} SDL_KeyboardDeviceEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_KeyboardEvent

Keyboard button event structure (event.key.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_KeyboardEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_KEY_DOWN or ::SDL_EVENT_KEY_UP
        */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window with keyboard focus, if any */
    SDL_KeyboardID which; /**< The keyboard instance id, or 0 if unknown
        or virtual */
    Uint8 state;      /**< ::SDL_PRESSED or ::SDL_RELEASED */
    Uint8 repeat;     /**< Non-zero if this is a key repeat */
    Uint8 padding2;
    Uint8 padding3;
    SDL_Keysym keysym; /**< The key that was pressed or released */
} SDL_KeyboardEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_KeyboardID

This is a unique ID for a keyboard for the time it is connected to the system, and is never reused for the lifetime of the application.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
typedef Uint32 SDL_KeyboardID;
```

Remarks

If the keyboard is disconnected and reconnected, it will get a new ID.

The ID value starts at 1 and increments from there. The value 0 is an invalid ID.

Version

This datatype is available since SDL 3.0.0.

SDL_Keycode

The SDL virtual key representation.

Header File

Defined in SDL3/SDL_keycode.h

Syntax

```
typedef Sint32 SDL_Keycode;

#define SDLK_SCANCODE_MASK (1<<30)
#define SDL_SCANCODE_TO_KEYCODE(X) (X | SDLK_SCANCODE_MASK)
#define SDLK_UNKNOWN  0
#define SDLK_RETURN '\r'
#define SDLK_ESCAPE '\x1B'
#define SDLK_BACKSPACE '\b'
#define SDLK_TAB  '\t'
#define SDLK_SPACE ' '
#define SDLK_EXCLAIM  '!'
#define SDLK_QUOTEDBL '"'
#define SDLK_HASH '#'
#define SDLK_PERCENT  '%'
#define SDLK_DOLLAR '$'
#define SDLK_AMPERSAND '&'
#define SDLK_QUOTE '\''
#define SDLK_LEFTPAREN '('
#define SDLK_RIGHTPAREN ')'
#define SDLK_ASTERISK '*'
#define SDLK_PLUS '+'
#define SDLK_COMMA ','
#define SDLK_MINUS '-'
#define SDLK_PERIOD '.'
#define SDLK_SLASH '/'
#define SDLK_0 '0'
#define SDLK_1 '1'
#define SDLK_2 '2'
#define SDLK_3 '3'
#define SDLK_4 '4'
#define SDLK_5 '5'
#define SDLK_6 '6'
#define SDLK_7 '7'
#define SDLK_8 '8'
#define SDLK_9 '9'
#define SDLK_COLON ':'
#define SDLK_SEMICOLON ';'
#define SDLK_LESS '<'
```

```
#define SDLK_EQUALS '='
#define SDLK_GREATER  '>'
#define SDLK_QUESTION '?'
#define SDLK_AT '@'
#define SDLK_LEFTBRACKET '['
#define SDLK_BACKSLASH '\\'
#define SDLK_RIGHTBRACKET ']'
#define SDLK_CARET '^'
#define SDLK_UNDERSCORE '_'
#define SDLK_BACKQUOTE '`'
#define SDLK_a 'a'
#define SDLK_b 'b'
#define SDLK_c 'c'
#define SDLK_d 'd'
#define SDLK_e 'e'
#define SDLK_f 'f'
#define SDLK_g 'g'
#define SDLK_h 'h'
#define SDLK_i 'i'
#define SDLK_j 'j'
#define SDLK_k 'k'
#define SDLK_l 'l'
#define SDLK_m 'm'
#define SDLK_n 'n'
#define SDLK_o 'o'
#define SDLK_p 'p'
#define SDLK_q 'q'
#define SDLK_r 'r'
#define SDLK_s 's'
#define SDLK_t 't'
#define SDLK_u 'u'
#define SDLK_v 'v'
#define SDLK_w 'w'
#define SDLK_x 'x'
#define SDLK_y 'y'
#define SDLK_z 'z'
#define SDLK_CAPSLOCK SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CAPSLOCK)
#define SDLK_F1 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F1)
#define SDLK_F2 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F2)
#define SDLK_F3 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F3)
#define SDLK_F4 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F4)
#define SDLK_F5 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F5)
#define SDLK_F6 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F6)
#define SDLK_F7 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F7)
#define SDLK_F8 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F8)
#define SDLK_F9 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F9)
#define SDLK_F10  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F10)
#define SDLK_F11  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F11)
#define SDLK_F12  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F12)
```

```
#define SDLK_PRINTSCREEN
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_PRINTSCREEN)
#define SDLK_SCROLLLOCK SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_SCROLLLOCK)
#define SDLK_PAUSE SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_PAUSE)
#define SDLK_INSERT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_INSERT)
#define SDLK_HOME SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_HOME)
#define SDLK_PAGEUP SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_PAGEUP)
#define SDLK_DELETE '\x7F'
#define SDLK_END  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_END)
#define SDLK_PAGEDOWN SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_PAGEDOWN)
#define SDLK_RIGHT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_RIGHT)
#define SDLK_LEFT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_LEFT)
#define SDLK_DOWN SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_DOWN)
#define SDLK_UP SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_UP)
#define SDLK_NUMLOCKCLEAR
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_NUMLOCKCLEAR)
#define SDLK_KP_DIVIDE SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_DIVIDE)
#define SDLK_KP_MULTIPLY
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_MULTIPLY)
#define SDLK_KP_MINUS SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_MINUS)
#define SDLK_KP_PLUS  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_PLUS)
#define SDLK_KP_ENTER SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_ENTER)
#define SDLK_KP_1 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_1)
#define SDLK_KP_2 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_2)
#define SDLK_KP_3 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_3)
#define SDLK_KP_4 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_4)
#define SDLK_KP_5 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_5)
#define SDLK_KP_6 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_6)
#define SDLK_KP_7 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_7)
#define SDLK_KP_8 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_8)
#define SDLK_KP_9 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_9)
#define SDLK_KP_0 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_0)
#define SDLK_KP_PERIOD SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_PERIOD)
#define SDLK_APPLICATION
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_APPLICATION)
#define SDLK_POWER SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_POWER)
#define SDLK_KP_EQUALS SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_EQUALS)
#define SDLK_F13  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F13)
#define SDLK_F14  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F14)
#define SDLK_F15  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F15)
#define SDLK_F16  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F16)
#define SDLK_F17  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F17)
#define SDLK_F18  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F18)
#define SDLK_F19  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F19)
#define SDLK_F20  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F20)
#define SDLK_F21  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F21)
#define SDLK_F22  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F22)
#define SDLK_F23  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F23)
#define SDLK_F24  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_F24)
#define SDLK_EXECUTE  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_EXECUTE)
```

```
#define SDLK_HELP SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_HELP)
#define SDLK_MENU SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_MENU)
#define SDLK_SELECT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_SELECT)
#define SDLK_STOP SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_STOP)
#define SDLK_AGAIN SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AGAIN)
#define SDLK_UNDO SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_UNDO)
#define SDLK_CUT  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CUT)
#define SDLK_COPY SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_COPY)
#define SDLK_PASTE SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_PASTE)
#define SDLK_FIND SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_FIND)
#define SDLK_MUTE SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_MUTE)
#define SDLK_VOLUMEUP SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_VOLUMEUP)
#define SDLK_VOLUMEDOWN SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_VOLUMEDOWN)
#define SDLK_KP_COMMA SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_COMMA)
#define SDLK_KP_EQUALSAS400
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_EQUALSAS400)
#define SDLK_ALTERASE SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_ALTERASE)
#define SDLK_SYSREQ SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_SYSREQ)
#define SDLK_CANCEL SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CANCEL)
#define SDLK_CLEAR SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CLEAR)
#define SDLK_PRIOR SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_PRIOR)
#define SDLK_RETURN2  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_RETURN2)
#define SDLK_SEPARATOR SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_SEPARATOR)
#define SDLK_OUT  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_OUT)
#define SDLK_OPER SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_OPER)
#define SDLK_CLEARAGAIN SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CLEARAGAIN)
#define SDLK_CRSEL SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CRSEL)
#define SDLK_EXSEL SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_EXSEL)
#define SDLK_KP_00 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_00)
#define SDLK_KP_000 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_000)
#define SDLK_THOUSANDSSEPARATOR
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_THOUSANDSSEPARATOR)
#define SDLK_DECIMALSEPARATOR
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_DECIMALSEPARATOR)
#define SDLK_CURRENCYUNIT
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CURRENCYUNIT)
#define SDLK_CURRENCYSUBUNIT
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CURRENCYSUBUNIT)
#define SDLK_KP_LEFTPAREN
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_LEFTPAREN)
#define SDLK_KP_RIGHTPAREN
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_RIGHTPAREN)
#define SDLK_KP_LEFTBRACE
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_LEFTBRACE)
#define SDLK_KP_RIGHTBRACE
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_RIGHTBRACE)
#define SDLK_KP_TAB SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_TAB)
#define SDLK_KP_BACKSPACE
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_BACKSPACE)
#define SDLK_KP_A SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_A)
```

```
#define SDLK_KP_B SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_B)
#define SDLK_KP_C SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_C)
#define SDLK_KP_D SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_D)
#define SDLK_KP_E SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_E)
#define SDLK_KP_F SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_F)
#define SDLK_KP_XOR SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_XOR)
#define SDLK_KP_POWER SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_POWER)
#define SDLK_KP_PERCENT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_PERCENT)
#define SDLK_KP_LESS  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_LESS)
#define SDLK_KP_GREATER SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_GREATER)
#define SDLK_KP_AMPERSAND
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_AMPERSAND)
#define SDLK_KP_DBLAMPERSAND
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_DBLAMPERSAND)
#define SDLK_KP_VERTICALBAR
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_VERTICALBAR)
#define SDLK_KP_DBLVERTICALBAR
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_DBLVERTICALBAR)
#define SDLK_KP_COLON SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_COLON)
#define SDLK_KP_HASH  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_HASH)
#define SDLK_KP_SPACE SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_SPACE)
#define SDLK_KP_AT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_AT)
#define SDLK_KP_EXCLAM SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_EXCLAM)
#define SDLK_KP_MEMSTORE
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_MEMSTORE)
#define SDLK_KP_MEMRECALL
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_MEMRECALL)
#define SDLK_KP_MEMCLEAR
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_MEMCLEAR)
#define SDLK_KP_MEMADD SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_MEMADD)
#define SDLK_KP_MEMSUBTRACT
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_MEMSUBTRACT)
#define SDLK_KP_MEMMULTIPLY
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_MEMMULTIPLY)
#define SDLK_KP_MEMDIVIDE
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_MEMDIVIDE)
#define SDLK_KP_PLUSMINUS
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_PLUSMINUS)
#define SDLK_KP_CLEAR SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_CLEAR)
#define SDLK_KP_CLEARENTRY
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_CLEARENTRY)
#define SDLK_KP_BINARY SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_BINARY)
#define SDLK_KP_OCTAL SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_OCTAL)
#define SDLK_KP_DECIMAL SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_DECIMAL)
#define SDLK_KP_HEXADECIMAL
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KP_HEXADECIMAL)
#define SDLK_LCTRL SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_LCTRL)
#define SDLK_LSHIFT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_LSHIFT)
#define SDLK_LALT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_LALT)
#define SDLK_LGUI SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_LGUI)
```

```
#define SDLK_RCTRL SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_RCTRL)
#define SDLK_RSHIFT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_RSHIFT)
#define SDLK_RALT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_RALT)
#define SDLK_RGUI SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_RGUI)
#define SDLK_MODE SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_MODE)
#define SDLK_AUDIONEXT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AUDIONEXT)
#define SDLK_AUDIOPREV SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AUDIOPREV)
#define SDLK_AUDIOSTOP SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AUDIOSTOP)
#define SDLK_AUDIOPLAY SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AUDIOPLAY)
#define SDLK_AUDIOMUTE SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AUDIOMUTE)
#define SDLK_MEDIASELECT
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_MEDIASELECT)
#define SDLK_WWW  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_WWW)
#define SDLK_MAIL SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_MAIL)
#define SDLK_CALCULATOR SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CALCULATOR)
#define SDLK_COMPUTER SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_COMPUTER)
#define SDLK_AC_SEARCH SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AC_SEARCH)
#define SDLK_AC_HOME  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AC_HOME)
#define SDLK_AC_BACK  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AC_BACK)
#define SDLK_AC_FORWARD SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AC_FORWARD)
#define SDLK_AC_STOP  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AC_STOP)
#define SDLK_AC_REFRESH SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AC_REFRESH)
#define SDLK_AC_BOOKMARKS
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AC_BOOKMARKS)
#define SDLK_BRIGHTNESSDOWN
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_BRIGHTNESSDOWN)
#define SDLK_BRIGHTNESSUP
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_BRIGHTNESSUP)
#define SDLK_DISPLAYSWITCH
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_DISPLAYSWITCH)
#define SDLK_KBDILLUMTOGGLE
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KBDILLUMTOGGLE)
#define SDLK_KBDILLUMDOWN
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KBDILLUMDOWN)
#define SDLK_KBDILLUMUP SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_KBDILLUMUP)
#define SDLK_EJECT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_EJECT)
#define SDLK_SLEEP SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_SLEEP)
#define SDLK_APP1 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_APP1)
#define SDLK_APP2 SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_APP2)
#define SDLK_AUDIOREWIND
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AUDIOREWIND)
#define SDLK_AUDIOFASTFORWARD
    SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_AUDIOFASTFORWARD)
#define SDLK_SOFTLEFT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_SOFTLEFT)
#define SDLK_SOFTRIGHT SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_SOFTRIGHT)
#define SDLK_CALL SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_CALL)
#define SDLK_ENDCALL  SDL_SCANCODE_TO_KEYCODE(SDL_SCANCODE_ENDCALL)
```

Remarks

Values of this type are used to represent keyboard keys using the current layout of the keyboard. These values include Unicode values representing the unmodified character that would be generated by pressing the key, or an `SDLK_*` constant for those keys that do not generate characters.

A special exception is the number keys at the top of the keyboard which map to SDLK_0...SDLK_9 on AZERTY layouts.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_KeyCode

SDL_Keymod

Enumeration of valid key mods (possibly OR'd together).

Header File

Defined in SDL3/SDL_keycode.h

Syntax

```
typedef enum SDL_Keymod
{
    SDL_KMOD_NONE = 0x0000, /**< no modifier is applicable. */
    SDL_KMOD_LSHIFT = 0x0001, /**< the left Shift key is down. */
    SDL_KMOD_RSHIFT = 0x0002, /**< the right Shift key is down. */
    SDL_KMOD_LCTRL = 0x0040, /**< the left Ctrl (Control) key is down. */
    SDL_KMOD_RCTRL = 0x0080, /**< the right Ctrl (Control) key is down.
        */
    SDL_KMOD_LALT = 0x0100, /**< the left Alt key is down. */
    SDL_KMOD_RALT = 0x0200, /**< the right Alt key is down. */
    SDL_KMOD_LGUI = 0x0400, /**< the left GUI key (often the Windows
        key) is down. */
    SDL_KMOD_RGUI = 0x0800, /**< the right GUI key (often the Windows
        key) is down. */
    SDL_KMOD_NUM = 0x1000,  /**< the Num Lock key (may be located on an
        extended keypad) is down. */
    SDL_KMOD_CAPS = 0x2000, /**< the Caps Lock key is down. */
    SDL_KMOD_MODE = 0x4000, /**< the !AltGr key is down. */
    SDL_KMOD_SCROLL = 0x8000, /**< the Scoll Lock key is down. */

    SDL_KMOD_CTRL = SDL_KMOD_LCTRL | SDL_KMOD_RCTRL, /**< Any Ctrl key
        is down. */
    SDL_KMOD_SHIFT = SDL_KMOD_LSHIFT | SDL_KMOD_RSHIFT, /**< Any Shift
        key is down. */
    SDL_KMOD_ALT = SDL_KMOD_LALT | SDL_KMOD_RALT,  /**< Any Alt key is
        down. */
    SDL_KMOD_GUI = SDL_KMOD_LGUI | SDL_KMOD_RGUI   /**< Any GUI key is
        down. */
} SDL_Keymod;
```

Version

This enum is available since SDL 3.0.0.

SDL_Keysym

The SDL keysym structure, used in key events.

Header File

Defined in SDL3/SDL_keyboard.h

Syntax

```
typedef struct SDL_Keysym
{
    SDL_Scancode scancode;   /**< SDL physical key code - see
        ::SDL_Scancode for details */
    SDL_Keycode sym;         /**< SDL virtual key code - see
        ::SDL_Keycode for details */
    Uint16 mod;              /**< current key modifiers */
    Uint32 unused;
} SDL_Keysym;
```

Remarks

If you are looking for translated character input, see the ::SDL_EVENT_TEXT_INPUT event.

Version

This struct is available since SDL 3.0.0.

SDL_Locale

A struct to provide locale data.

Header File

Defined in SDL3/SDL_locale.h

Syntax

```
typedef struct SDL_Locale
{
    const char *language; /**< A language name, like "en" for English. */
    const char *country; /**< A country, like "US" for America. Can be
        NULL. */
} SDL_Locale;
```

Remarks

Locale data is split into a spoken language, like English, and an optional country, like Canada. The language will be in ISO-639 format (so English would be "en"), and the country, if not NULL, will be an ISO-3166 country code (so Canada would be "CA").

Version

This function is available since SDL 3.0.0.

See Also

- SDL_GetPreferredLocales

SDL_LogCategory

The predefined log categories

Header File

Defined in SDL3/SDL_log.h

Syntax

```
typedef enum SDL_LogCategory
{
    SDL_LOG_CATEGORY_APPLICATION,
    SDL_LOG_CATEGORY_ERROR,
    SDL_LOG_CATEGORY_ASSERT,
    SDL_LOG_CATEGORY_SYSTEM,
    SDL_LOG_CATEGORY_AUDIO,
    SDL_LOG_CATEGORY_VIDEO,
    SDL_LOG_CATEGORY_RENDER,
    SDL_LOG_CATEGORY_INPUT,
    SDL_LOG_CATEGORY_TEST,

    /* Reserved for future SDL library use */
    SDL_LOG_CATEGORY_RESERVED1,
    SDL_LOG_CATEGORY_RESERVED2,
    SDL_LOG_CATEGORY_RESERVED3,
    SDL_LOG_CATEGORY_RESERVED4,
    SDL_LOG_CATEGORY_RESERVED5,
    SDL_LOG_CATEGORY_RESERVED6,
    SDL_LOG_CATEGORY_RESERVED7,
    SDL_LOG_CATEGORY_RESERVED8,
    SDL_LOG_CATEGORY_RESERVED9,
    SDL_LOG_CATEGORY_RESERVED10,

    /* Beyond this point is reserved for application use, e.g.
       enum {
           MYAPP_CATEGORY_AWESOME1 = SDL_LOG_CATEGORY_CUSTOM,
           MYAPP_CATEGORY_AWESOME2,
           MYAPP_CATEGORY_AWESOME3,
           ...
       };
     */
    SDL_LOG_CATEGORY_CUSTOM
} SDL_LogCategory;
```

Remarks

By default the application category is enabled at the INFO level, the assert category is enabled at the WARN level, test is enabled at the VERBOSE level and all other categories are enabled at the ERROR level.

Version

This enum is available since SDL 3.0.0.

SDL_LogOutputFunction

The prototype for the log output callback function.

Header File

Defined in SDL3/SDL_log.h

Syntax

```
typedef void (SDLCALL *SDL_LogOutputFunction)(void *userdata, int
    category, SDL_LogPriority priority, const char *message);
```

Function Parameters

userdata	what was passed as `userdata` to SDL_SetLogOutputFunction()
category	the category of the message
priority	the priority of the message
message	the message being output

Remarks

This function is called by SDL when there is new text to be logged.

Version

This datatype is available since SDL 3.0.0.

SDL_LogPriority

The predefined log priorities

Header File

Defined in SDL3/SDL_log.h

Syntax

```
typedef enum SDL_LogPriority
{
    SDL_LOG_PRIORITY_VERBOSE = 1,
    SDL_LOG_PRIORITY_DEBUG,
    SDL_LOG_PRIORITY_INFO,
    SDL_LOG_PRIORITY_WARN,
    SDL_LOG_PRIORITY_ERROR,
    SDL_LOG_PRIORITY_CRITICAL,
    SDL_NUM_LOG_PRIORITIES
} SDL_LogPriority;
```

Version

This enum is available since SDL 3.0.0.

SDL_main_func

The prototype for the application's main() function

Header File

Defined in SDL3/SDL_main.h

Syntax

```
typedef int (SDLCALL *SDL_main_func)(int argc, char *argv[]);
```

Version

This datatype is available since SDL 3.0.0.

SDL_MatrixCoefficients

The matrix coefficients.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef enum SDL_MatrixCoefficients
{
    SDL_MATRIX_COEFFICIENTS_IDENTITY = 0,
    SDL_MATRIX_COEFFICIENTS_BT709 = 1,          /**< ITU-R BT.709-6 */
    SDL_MATRIX_COEFFICIENTS_UNSPECIFIED = 2,
    SDL_MATRIX_COEFFICIENTS_FCC = 4,            /**< US FCC */
    SDL_MATRIX_COEFFICIENTS_BT470BG = 5,        /**< ITU-R BT.470-6
        System B, G / ITU-R BT.601-7 625, functionally the same as
        SDL_MATRIX_COEFFICIENTS_BT601 */
    SDL_MATRIX_COEFFICIENTS_BT601 = 6,          /**< ITU-R BT.601-7 525 */
    SDL_MATRIX_COEFFICIENTS_SMPTE240 = 7,       /**< SMPTE 240M */
    SDL_MATRIX_COEFFICIENTS_YCGCO = 8,
    SDL_MATRIX_COEFFICIENTS_BT2020_NCL = 9,     /**< ITU-R BT.2020-2
        non-constant luminance */
    SDL_MATRIX_COEFFICIENTS_BT2020_CL = 10,     /**< ITU-R BT.2020-2
        constant luminance */
    SDL_MATRIX_COEFFICIENTS_SMPTE2085 = 11,     /**< SMPTE ST 2085 */
    SDL_MATRIX_COEFFICIENTS_CHROMA_DERIVED_NCL = 12,
    SDL_MATRIX_COEFFICIENTS_CHROMA_DERIVED_CL = 13,
    SDL_MATRIX_COEFFICIENTS_ICTCP = 14,         /**< ITU-R BT.2100-0
        ICTCP */
    SDL_MATRIX_COEFFICIENTS_CUSTOM = 31
} SDL_MatrixCoefficients;
```

Remarks

These are as described by https://www.itu.int/rec/T-REC-H.273-201612-S/en

Version

This enum is available since SDL 3.0.0.

SDL_MessageBoxButtonData

Individual button data.

Header File

Defined in SDL3/SDL_messagebox.h

Syntax

```
typedef struct SDL_MessageBoxButtonData
{
   Uint32 flags;     /**< ::SDL_MessageBoxButtonFlags */
   int buttonID;     /**< User defined button id (value returned via
       SDL_ShowMessageBox) */
   const char *text; /**< The UTF-8 button text */
} SDL_MessageBoxButtonData;
```

Version

This struct is available since SDL 3.0.0.

SDL_MessageBoxButtonFlags

Flags for SDL_MessageBoxButtonData.

Header File

Defined in SDL3/SDL_messagebox.h

Syntax

```
typedef enum SDL_MessageBoxButtonFlags
{
    SDL_MESSAGEBOX_BUTTON_RETURNKEY_DEFAULT = 0x00000001, /**< Marks the
        default button when return is hit */
    SDL_MESSAGEBOX_BUTTON_ESCAPEKEY_DEFAULT = 0x00000002 /**< Marks the
        default button when escape is hit */
} SDL_MessageBoxButtonFlags;
```

Version

This enum is available since SDL 3.0.0.

SDL_MessageBoxColor

RGB value used in a message box color scheme

Header File

Defined in SDL3/SDL_messagebox.h

Syntax

```
typedef struct SDL_MessageBoxColor
{
   Uint8 r, g, b;
} SDL_MessageBoxColor;
```

Version

This struct is available since SDL 3.0.0.

SDL_MessageBoxColorScheme

A set of colors to use for message box dialogs

Header File

Defined in SDL3/SDL_messagebox.h

Syntax

```
typedef struct SDL_MessageBoxColorScheme
{
    SDL_MessageBoxColor colors[SDL_MESSAGEBOX_COLOR_MAX];
} SDL_MessageBoxColorScheme;
```

Version

This struct is available since SDL 3.0.0.

SDL_MessageBoxColorType

An enumeration of indices inside the colors array of SDL_MessageBoxColorScheme.

Header File

Defined in SDL3/SDL_messagebox.h

Syntax

```
typedef enum SDL_MessageBoxColorType
{
    SDL_MESSAGEBOX_COLOR_BACKGROUND,
    SDL_MESSAGEBOX_COLOR_TEXT,
    SDL_MESSAGEBOX_COLOR_BUTTON_BORDER,
    SDL_MESSAGEBOX_COLOR_BUTTON_BACKGROUND,
    SDL_MESSAGEBOX_COLOR_BUTTON_SELECTED,
    SDL_MESSAGEBOX_COLOR_MAX                /**< Size of the colors
        array of SDL_MessageBoxColorScheme. */
} SDL_MessageBoxColorType;
```

SDL_MessageBoxData

MessageBox structure containing title, text, window, etc.

Header File

Defined in SDL3/SDL_messagebox.h

Syntax

```
typedef struct SDL_MessageBoxData
{
    Uint32 flags;                   /**< ::SDL_MessageBoxFlags */
    SDL_Window *window;             /**< Parent window, can be NULL */
    const char *title;              /**< UTF-8 title */
    const char *message;            /**< UTF-8 message text */

    int numbuttons;
    const SDL_MessageBoxButtonData *buttons;

    const SDL_MessageBoxColorScheme *colorScheme; /**<
        ::SDL_MessageBoxColorScheme, can be NULL to use system settings
        */
} SDL_MessageBoxData;
```

Version

This struct is available since SDL 3.0.0.

SDL_MessageBoxFlags

SDL_MessageBox flags.

Header File

Defined in SDL3/SDL_messagebox.h

Syntax

```
typedef enum SDL_MessageBoxFlags
{
    SDL_MESSAGEBOX_ERROR             = 0x00000010, /**< error dialog */
    SDL_MESSAGEBOX_WARNING           = 0x00000020, /**< warning dialog
        */
    SDL_MESSAGEBOX_INFORMATION       = 0x00000040, /**< informational
        dialog */
    SDL_MESSAGEBOX_BUTTONS_LEFT_TO_RIGHT = 0x00000080, /**< buttons
        placed left to right */
    SDL_MESSAGEBOX_BUTTONS_RIGHT_TO_LEFT = 0x00000100 /**< buttons
        placed right to left */
} SDL_MessageBoxFlags;
```

Remarks

If supported will display warning icon, etc.

Version

This enum is available since SDL 3.0.0.

SDL_MetalView

A handle to a CAMetalLayer-backed NSView (macOS) or UIView (iOS/tvOS).

Header File

Defined in SDL3/SDL_metal.h

Syntax

```
typedef void *SDL_MetalView;
```

Version

This datatype is available since SDL 3.0.0.

SDL_MouseButtonEvent

Mouse button event structure (event.button.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_MouseButtonEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_MOUSE_BUTTON_DOWN or
        ::SDL_EVENT_MOUSE_BUTTON_UP */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window with mouse focus, if any */
    SDL_MouseID which; /**< The mouse instance id, SDL_TOUCH_MOUSEID, or
        SDL_PEN_MOUSEID */
    Uint8 button;     /**< The mouse button index */
    Uint8 state;      /**< ::SDL_PRESSED or ::SDL_RELEASED */
    Uint8 clicks;     /**< 1 for single-click, 2 for double-click, etc.
        */
    Uint8 padding;
    float x;          /**< X coordinate, relative to window */
    float y;          /**< Y coordinate, relative to window */
} SDL_MouseButtonEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_MouseDeviceEvent

Mouse device event structure (event.mdevice.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_MouseDeviceEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_MOUSE_ADDED or
        ::SDL_EVENT_MOUSE_REMOVED */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_MouseID which; /**< The mouse instance id */
} SDL_MouseDeviceEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_MouseMotionEvent

Mouse motion event structure (event.motion.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_MouseMotionEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_MOUSE_MOTION */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window with mouse focus, if any */
    SDL_MouseID which; /**< The mouse instance id, SDL_TOUCH_MOUSEID, or
        SDL_PEN_MOUSEID */
    Uint32 state;     /**< The current button state */
    float x;          /**< X coordinate, relative to window */
    float y;          /**< Y coordinate, relative to window */
    float xrel;       /**< The relative motion in the X direction */
    float yrel;       /**< The relative motion in the Y direction */
} SDL_MouseMotionEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_MouseWheelDirection

Scroll direction types for the Scroll event

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
typedef enum SDL_MouseWheelDirection
{
    SDL_MOUSEWHEEL_NORMAL,  /**< The scroll direction is normal */
    SDL_MOUSEWHEEL_FLIPPED  /**< The scroll direction is flipped /
        natural */
} SDL_MouseWheelDirection;
```

Version

This enum is available since SDL 3.0.0.

SDL_MouseWheelEvent

Mouse wheel event structure (event.wheel.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_MouseWheelEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_MOUSE_WHEEL */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window with mouse focus, if any */
    SDL_MouseID which; /**< The mouse instance id, SDL_TOUCH_MOUSEID, or
        SDL_PEN_MOUSEID */
    float x;          /**< The amount scrolled horizontally, positive to
        the right and negative to the left */
    float y;          /**< The amount scrolled vertically, positive away
        from the user and negative toward the user */
    SDL_MouseWheelDirection direction; /**< Set to one of the
        SDL_MOUSEWHEEL_* defines. When FLIPPED the values in X and Y
        will be opposite. Multiply by -1 to change them back */
    float mouse_x;    /**< X coordinate, relative to window */
    float mouse_y;    /**< Y coordinate, relative to window */
} SDL_MouseWheelEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_Mutex

A means to serialize access to a resource between threads.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
typedef struct SDL_Mutex SDL_Mutex;
```

Remarks

Mutexes (short for "mutual exclusion") are a synchronization primitive that allows exactly one thread to proceed at a time.

Wikipedia has a thorough explanation of the concept:
https://en.wikipedia.org/wiki/Mutex

Version

This struct is available since SDL 3.0.0.

SDL_Palette

A set of indexed colors representing a palette.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef struct SDL_Palette
{
    int ncolors;      /**< number of elements in ‘colors‘. */
    SDL_Color *colors; /**< an array of colors, ‘ncolors‘ long. */
    Uint32 version;   /**< internal use only, do not touch. */
    int refcount;     /**< internal use only, do not touch. */
} SDL_Palette;
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_PixelFormat
- SDL_SetPaletteColors

SDL_PenAxis

Pen axis indices

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
typedef enum SDL_PenAxis
{
    SDL_PEN_AXIS_PRESSURE = 0,          /**< Pen pressure.
        Unidirectional: 0..1.0 */
    SDL_PEN_AXIS_XTILT,                 /**< Pen horizontal tilt angle.
        Bidirectional: -90.0..90.0 (left-to-right).
                                            The physical max/min tilt
                                                may be smaller than
                                                -90.0 / 90.0, cf.
                                                SDL_PenCapabilityInfo
                                                */
    SDL_PEN_AXIS_YTILT,                 /**< Pen vertical tilt angle.
        Bidirectional: -90.0..90.0 (top-to-down).
                                            The physical max/min tilt
                                                may be smaller than
                                                -90.0 / 90.0, cf.
                                                SDL_PenCapabilityInfo
                                                */
    SDL_PEN_AXIS_DISTANCE,              /**< Pen distance to drawing
        surface. Unidirectional: 0.0..1.0 */
    SDL_PEN_AXIS_ROTATION,              /**< Pen barrel rotation.
        Bidirectional: -180..179.9 (clockwise, 0 is facing up, -180.0 is
        facing down). */
    SDL_PEN_AXIS_SLIDER,                /**< Pen finger wheel or slider
        (e.g., Airbrush Pen). Unidirectional: 0..1.0 */
    SDL_PEN_NUM_AXES,                   /**< Last valid axis index */
    SDL_PEN_AXIS_LAST = SDL_PEN_NUM_AXES - 1 /**< Last axis index plus 1
        */
} SDL_PenAxis;
```

Remarks

Below are the valid indices to the "axis" array from SDL_PenMotionEvent and SDL_PenButtonEvent. The axis indices form a contiguous range of ints from 0 to SDL_PEN_AXIS_LAST, inclusive. All "axis[]" entries are either normalised to 0..1 or report a (positive or negative) angle in degrees, with 0.0 representing

the centre. Not all pens/backends support all axes: unsupported entries are always "0.0f".

To convert angles for tilt and rotation into vector representation, use SDL_sinf on the XTILT, YTILT, or ROTATION component, for example: `SDL_sinf(xtilt * SDL_PI_F / 180.0)`.

Version

This enum is available since SDL 3.0.0

SDL_PenButtonEvent

Pressure-sensitive pen button event structure (event.pbutton.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_PenButtonEvent
{
    SDL_EventType type;   /**< ::SDL_EVENT_PEN_BUTTON_DOWN or
        ::SDL_EVENT_PEN_BUTTON_UP */
    Uint32 reserved;
    Uint64 timestamp;     /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window with pen focus, if any */
    SDL_PenID which;      /**< The pen instance id */
    Uint8 button;         /**< The pen button index (1 represents the
        pen tip for compatibility with mouse events) */
    Uint8 state;          /**< ::SDL_PRESSED or ::SDL_RELEASED */
    Uint16 pen_state;     /**< Pen button masks (where SDL_BUTTON(1) is
        the first button, SDL_BUTTON(2) is the second button etc.),
        ::SDL_PEN_DOWN_MASK is set if the pen is touching the surface,
        and ::SDL_PEN_ERASER_MASK is set if the pen is (used as) an
        eraser. */
    float x;              /**< X coordinate, relative to window */
    float y;              /**< Y coordinate, relative to window */
    float axes[SDL_PEN_NUM_AXES]; /**< Pen axes such as pressure and
        tilt (ordered as per ::SDL_PenAxis) */
} SDL_PenButtonEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_PenCapabilityInfo

Pen capabilities, as reported by SDL_GetPenCapabilities()

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
typedef struct SDL_PenCapabilityInfo
{
    float max_tilt;  /**< Physical maximum tilt angle, for XTILT and
        YTILT, or SDL_PEN_INFO_UNKNOWN . Pens cannot typically tilt all
        the way to 90 degrees, so this value is usually less than 90.0.
        */
    Uint32 wacom_id; /**< For Wacom devices: wacom tool type ID,
        otherwise 0 (useful e.g. with libwacom) */
    Sint8 num_buttons; /**< Number of pen buttons (not counting the pen
        tip), or SDL_PEN_INFO_UNKNOWN */
} SDL_PenCapabilityInfo;
```

Version

This struct is available since SDL 3.0.0.

SDL_PenMotionEvent

Pressure-sensitive pen motion / pressure / angle event structure (event.pmotion.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_PenMotionEvent
{
    SDL_EventType type;   /**< ::SDL_EVENT_PEN_MOTION */
    Uint32 reserved;
    Uint64 timestamp;     /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window with pen focus, if any */
    SDL_PenID which;      /**< The pen instance id */
    Uint8 padding1;
    Uint8 padding2;
    Uint16 pen_state;     /**< Pen button masks (where SDL_BUTTON(1) is
        the first button, SDL_BUTTON(2) is the second button etc.),
        ::SDL_PEN_DOWN_MASK is set if the pen is touching the surface,
        and ::SDL_PEN_ERASER_MASK is set if the pen is (used as) an
        eraser. */
    float x;              /**< X coordinate, relative to window */
    float y;              /**< Y coordinate, relative to window */
    float axes[SDL_PEN_NUM_AXES]; /**< Pen axes such as pressure and
        tilt (ordered as per ::SDL_PenAxis) */
} SDL_PenMotionEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_PenSubtype

Pen types

Header File

Defined in SDL3/SDL_pen.h

Syntax

```
typedef enum SDL_PenSubtype
{
    SDL_PEN_TYPE_UNKNOWN = 0,
    SDL_PEN_TYPE_ERASER = 1,              /**< Eraser */
    SDL_PEN_TYPE_PEN,                     /**< Generic pen; this is the
        default. */
    SDL_PEN_TYPE_PENCIL,                  /**< Pencil */
    SDL_PEN_TYPE_BRUSH,                   /**< Brush-like device */
    SDL_PEN_TYPE_AIRBRUSH,                /**< Airbrush device that
        "sprays" ink */
    SDL_PEN_TYPE_LAST = SDL_PEN_TYPE_AIRBRUSH /**< Last valid pen type */
} SDL_PenSubtype;
```

Remarks

Some pens identify as a particular type of drawing device (e.g., an airbrush or a pencil).

Version

This enum is available since SDL 3.0.0

SDL_PenTipEvent

Pressure-sensitive pen touched or stopped touching surface (event.ptip.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_PenTipEvent
{
    SDL_EventType type;   /**< ::SDL_EVENT_PEN_DOWN or
        ::SDL_EVENT_PEN_UP */
    Uint32 reserved;
    Uint64 timestamp;     /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window with pen focus, if any */
    SDL_PenID which;      /**< The pen instance id */
    Uint8 tip;            /**< ::SDL_PEN_TIP_INK when using a regular
        pen tip, or ::SDL_PEN_TIP_ERASER if the pen is being used as an
        eraser (e.g., flipped to use the eraser tip) */
    Uint8 state;          /**< ::SDL_PRESSED on ::SDL_EVENT_PEN_DOWN and
        ::SDL_RELEASED on ::SDL_EVENT_PEN_UP */
    Uint16 pen_state;     /**< Pen button masks (where SDL_BUTTON(1) is
        the first button, SDL_BUTTON(2) is the second button etc.),
        ::SDL_PEN_DOWN_MASK is set if the pen is touching the surface,
        and ::SDL_PEN_ERASER_MASK is set if the pen is (used as) an
        eraser. */
    float x;              /**< X coordinate, relative to window */
    float y;              /**< Y coordinate, relative to window */
    float axes[SDL_PEN_NUM_AXES]; /**< Pen axes such as pressure and
        tilt (ordered as per ::SDL_PenAxis) */
} SDL_PenTipEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_PixelFormat

Details about the format of a pixel.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef struct SDL_PixelFormat
{
    SDL_PixelFormatEnum format;
    SDL_Palette *palette;
    Uint8 bits_per_pixel;
    Uint8 bytes_per_pixel;
    Uint8 padding[2];
    Uint32 Rmask;
    Uint32 Gmask;
    Uint32 Bmask;
    Uint32 Amask;
    Uint8 Rloss;
    Uint8 Gloss;
    Uint8 Bloss;
    Uint8 Aloss;
    Uint8 Rshift;
    Uint8 Gshift;
    Uint8 Bshift;
    Uint8 Ashift;
    int refcount;
    struct SDL_PixelFormat *next;
} SDL_PixelFormat;
```

Remarks

Generally this is used with SDL_Surface, and covers many possible configurations, including paletted data and various bit patterns.

Version

This struct is available since SDL 3.0.0.

SDL_Point

The structure that defines a point (using integers).

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
typedef struct SDL_Point
{
    int x;
    int y;
} SDL_Point;
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_GetRectEnclosingPoints
- SDL_PointInRect

SDL_PowerState

The basic state for the system's power supply.

Header File

Defined in SDL3/SDL_power.h

Syntax

```
typedef enum SDL_PowerState
{
    SDL_POWERSTATE_ERROR = -1, /**< error determining power status */
    SDL_POWERSTATE_UNKNOWN,   /**< cannot determine power status */
    SDL_POWERSTATE_ON_BATTERY, /**< Not plugged in, running on the
        battery */
    SDL_POWERSTATE_NO_BATTERY, /**< Plugged in, no battery available */
    SDL_POWERSTATE_CHARGING,  /**< Plugged in, charging battery */
    SDL_POWERSTATE_CHARGED    /**< Plugged in, battery charged */
} SDL_PowerState;
```

Remarks

These are results returned by SDL_GetPowerInfo().

Code Examples

```
if (SDL_GetPowerInfo(NULL, NULL) == SDL_POWERSTATE_ON_BATTERY) {
    SDL_Log("You should plug in your laptop before running this
        update.");
}
```

Version

This enum is available since SDL 3.0.0

SDL_PropertiesID

SDL properties ID

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
typedef Uint32 SDL_PropertiesID;
```

Version

This datatype is available since SDL 3.0.0.

SDL_PropertyType

SDL property type

Header File

Defined in SDL3/SDL_properties.h

Syntax

```
typedef enum SDL_PropertyType
{
    SDL_PROPERTY_TYPE_INVALID,
    SDL_PROPERTY_TYPE_POINTER,
    SDL_PROPERTY_TYPE_STRING,
    SDL_PROPERTY_TYPE_NUMBER,
    SDL_PROPERTY_TYPE_FLOAT,
    SDL_PROPERTY_TYPE_BOOLEAN
} SDL_PropertyType;
```

Version

This enum is available since SDL 3.0.0.

SDL_QuitEvent

The "quit requested" event

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_QuitEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_QUIT */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
} SDL_QuitEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_Rect

A rectangle, with the origin at the upper left (using integers).

Header File

Defined in SDL3/SDL_rect.h

Syntax

```
typedef struct SDL_Rect
{
    int x, y;
    int w, h;
} SDL_Rect;
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_RectEmpty
- SDL_RectsEqual
- SDL_HasRectIntersection
- SDL_GetRectIntersection
- SDL_GetRectAndLineIntersection
- SDL_GetRectUnion
- SDL_GetRectEnclosingPoints

SDL_Renderer

A structure representing rendering state

Header File

Defined in SDL3/SDL_render.h

Syntax

```
struct SDL_Renderer;
```

Version

This struct is available since SDL 3.0.0.

SDL_RendererFlags

Flags used when creating a rendering context.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
typedef enum SDL_RendererFlags
{
    SDL_RENDERER_PRESENTVSYNC = 0x00000004 /**< Present is synchronized
        with the refresh rate */
} SDL_RendererFlags;
```

Version

This enum is available since SDL 3.0.0.

SDL_RendererInfo

Information on the capabilities of a render driver or context.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
typedef struct SDL_RendererInfo
{
    const char *name;        /**< The name of the renderer */
    Uint32 flags;            /**< Supported ::SDL_RendererFlags */
    int num_texture_formats; /**< The number of available texture
        formats */
    SDL_PixelFormatEnum texture_formats[16]; /**< The available texture
        formats */
    int max_texture_width;   /**< The maximum texture width */
    int max_texture_height;  /**< The maximum texture height */
} SDL_RendererInfo;
```

Version

This struct is available since SDL 3.0.0.

SDL_RendererLogicalPresentation

How the logical size is mapped to the output.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
typedef enum SDL_RendererLogicalPresentation
{
    SDL_LOGICAL_PRESENTATION_DISABLED, /**< There is no logical size in
        effect */
    SDL_LOGICAL_PRESENTATION_STRETCH, /**< The rendered content is
        stretched to the output resolution */
    SDL_LOGICAL_PRESENTATION_LETTERBOX, /**< The rendered content is fit
        to the largest dimension and the other dimension is letterboxed
        with black bars */
    SDL_LOGICAL_PRESENTATION_OVERSCAN, /**< The rendered content is fit
        to the smallest dimension and the other dimension extends beyond
        the output bounds */
    SDL_LOGICAL_PRESENTATION_INTEGER_SCALE /**< The rendered content is
        scaled up by integer multiples to fit the output resolution */
} SDL_RendererLogicalPresentation;
```

Version

This enum is available since SDL 3.0.0.

SDL_RWLock

A mutex that allows read-only threads to run in parallel.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
typedef struct SDL_RWLock SDL_RWLock;
```

Remarks

A rwlock is roughly the same concept as SDL_Mutex, but allows threads that request read-only access to all hold the lock at the same time. If a thread requests write access, it will block until all read-only threads have released the lock, and no one else can hold the thread (for reading or writing) at the same time as the writing thread.

This can be more efficient in cases where several threads need to access data frequently, but changes to that data are rare.

There are other rules that apply to rwlocks that don't apply to mutexes, about how threads are scheduled and when they can be recursively locked. These are documented in the other rwlock functions.

Version

This struct is available since SDL 3.0.0.

SDL_ScaleMode

The scaling mode.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
typedef enum SDL_ScaleMode
{
    SDL_SCALEMODE_NEAREST, /**< nearest pixel sampling */
    SDL_SCALEMODE_LINEAR, /**< linear filtering */
    SDL_SCALEMODE_BEST   /**< anisotropic filtering */
} SDL_ScaleMode;
```

Version

This enum is available since SDL 3.0.0.

SDL_Scancode

The SDL keyboard scancode representation.

Header File

Defined in SDL3/SDL_scancode.h

Syntax

```
typedef enum SDL_Scancode
{
    SDL_SCANCODE_UNKNOWN = 0,

    /**
     *  \name Usage page 0x07
     *
     *  These values are from usage page 0x07 (USB keyboard page).
     */
    /* @{ */

    SDL_SCANCODE_A = 4,
    SDL_SCANCODE_B = 5,
    SDL_SCANCODE_C = 6,
    SDL_SCANCODE_D = 7,
    SDL_SCANCODE_E = 8,
    SDL_SCANCODE_F = 9,
    SDL_SCANCODE_G = 10,
    SDL_SCANCODE_H = 11,
    SDL_SCANCODE_I = 12,
    SDL_SCANCODE_J = 13,
    SDL_SCANCODE_K = 14,
    SDL_SCANCODE_L = 15,
    SDL_SCANCODE_M = 16,
    SDL_SCANCODE_N = 17,
    SDL_SCANCODE_O = 18,
    SDL_SCANCODE_P = 19,
    SDL_SCANCODE_Q = 20,
    SDL_SCANCODE_R = 21,
    SDL_SCANCODE_S = 22,
    SDL_SCANCODE_T = 23,
    SDL_SCANCODE_U = 24,
    SDL_SCANCODE_V = 25,
    SDL_SCANCODE_W = 26,
    SDL_SCANCODE_X = 27,
    SDL_SCANCODE_Y = 28,
    SDL_SCANCODE_Z = 29,
```

```
SDL_SCANCODE_1 = 30,
SDL_SCANCODE_2 = 31,
SDL_SCANCODE_3 = 32,
SDL_SCANCODE_4 = 33,
SDL_SCANCODE_5 = 34,
SDL_SCANCODE_6 = 35,
SDL_SCANCODE_7 = 36,
SDL_SCANCODE_8 = 37,
SDL_SCANCODE_9 = 38,
SDL_SCANCODE_0 = 39,

SDL_SCANCODE_RETURN = 40,
SDL_SCANCODE_ESCAPE = 41,
SDL_SCANCODE_BACKSPACE = 42,
SDL_SCANCODE_TAB = 43,
SDL_SCANCODE_SPACE = 44,

SDL_SCANCODE_MINUS = 45,
SDL_SCANCODE_EQUALS = 46,
SDL_SCANCODE_LEFTBRACKET = 47,
SDL_SCANCODE_RIGHTBRACKET = 48,
SDL_SCANCODE_BACKSLASH = 49, /**< Located at the lower left of the
    return
                          *  key on ISO keyboards and at the right
                              end
                          *  of the QWERTY row on ANSI keyboards.
                          *  Produces REVERSE SOLIDUS (backslash)
                              and
                          *  VERTICAL LINE in a US layout, REVERSE
                          *  SOLIDUS and VERTICAL LINE in a UK Mac
                          *  layout, NUMBER SIGN and TILDE in a UK
                          *  Windows layout, DOLLAR SIGN and POUND
                              SIGN
                          *  in a Swiss German layout, NUMBER SIGN
                              and
                          *  APOSTROPHE in a German layout, GRAVE
                          *  ACCENT and POUND SIGN in a French Mac
                          *  layout, and ASTERISK and MICRO SIGN
                              in a
                          *  French Windows layout.
                          */
SDL_SCANCODE_NONUSHASH = 50, /**< ISO USB keyboards actually use
    this code
                          *  instead of 49 for the same key, but
                              all
                          *  OSes I've seen treat the two codes
                          *  identically. So, as an implementor,
                              unless
                          *  your keyboard generates both of those
```

```
                            *   codes and your OS treats them
                                differently,
                            *   you should generate
                                SDL_SCANCODE_BACKSLASH
                            *   instead of this code. As a user, you
                            *   should not rely on this code because
                                SDL
                            *   will never generate it with most
                                (all?)
                            *   keyboards.
                            */
SDL_SCANCODE_SEMICOLON = 51,
SDL_SCANCODE_APOSTROPHE = 52,
SDL_SCANCODE_GRAVE = 53, /**< Located in the top left corner (on
    both ANSI
                        *   and ISO keyboards). Produces GRAVE ACCENT
                            and
                        *   TILDE in a US Windows layout and in US
                            and UK
                        *   Mac layouts on ANSI keyboards, GRAVE
                            ACCENT
                        *   and NOT SIGN in a UK Windows layout,
                            SECTION
                        *   SIGN and PLUS-MINUS SIGN in US and UK Mac
                        *   layouts on ISO keyboards, SECTION SIGN and
                        *   DEGREE SIGN in a Swiss German layout (Mac:
                        *   only on ISO keyboards), CIRCUMFLEX ACCENT
                            and
                        *   DEGREE SIGN in a German layout (Mac: only
                            on
                        *   ISO keyboards), SUPERSCRIPT TWO and TILDE
                            in a
                        *   French Windows layout, COMMERCIAL AT and
                        *   NUMBER SIGN in a French Mac layout on ISO
                        *   keyboards, and LESS-THAN SIGN and
                            GREATER-THAN
                        *   SIGN in a Swiss German, German, or French
                            Mac
                        *   layout on ANSI keyboards.
                        */
SDL_SCANCODE_COMMA = 54,
SDL_SCANCODE_PERIOD = 55,
SDL_SCANCODE_SLASH = 56,

SDL_SCANCODE_CAPSLOCK = 57,

SDL_SCANCODE_F1 = 58,
SDL_SCANCODE_F2 = 59,
SDL_SCANCODE_F3 = 60,
SDL_SCANCODE_F4 = 61,
```

```
SDL_SCANCODE_F5 = 62,
SDL_SCANCODE_F6 = 63,
SDL_SCANCODE_F7 = 64,
SDL_SCANCODE_F8 = 65,
SDL_SCANCODE_F9 = 66,
SDL_SCANCODE_F10 = 67,
SDL_SCANCODE_F11 = 68,
SDL_SCANCODE_F12 = 69,

SDL_SCANCODE_PRINTSCREEN = 70,
SDL_SCANCODE_SCROLLLOCK = 71,
SDL_SCANCODE_PAUSE = 72,
SDL_SCANCODE_INSERT = 73, /**< insert on PC, help on some Mac
    keyboards (but
                           does send code 73, not 117) */
SDL_SCANCODE_HOME = 74,
SDL_SCANCODE_PAGEUP = 75,
SDL_SCANCODE_DELETE = 76,
SDL_SCANCODE_END = 77,
SDL_SCANCODE_PAGEDOWN = 78,
SDL_SCANCODE_RIGHT = 79,
SDL_SCANCODE_LEFT = 80,
SDL_SCANCODE_DOWN = 81,
SDL_SCANCODE_UP = 82,

SDL_SCANCODE_NUMLOCKCLEAR = 83, /**< num lock on PC, clear on Mac
    keyboards
                             */
SDL_SCANCODE_KP_DIVIDE = 84,
SDL_SCANCODE_KP_MULTIPLY = 85,
SDL_SCANCODE_KP_MINUS = 86,
SDL_SCANCODE_KP_PLUS = 87,
SDL_SCANCODE_KP_ENTER = 88,
SDL_SCANCODE_KP_1 = 89,
SDL_SCANCODE_KP_2 = 90,
SDL_SCANCODE_KP_3 = 91,
SDL_SCANCODE_KP_4 = 92,
SDL_SCANCODE_KP_5 = 93,
SDL_SCANCODE_KP_6 = 94,
SDL_SCANCODE_KP_7 = 95,
SDL_SCANCODE_KP_8 = 96,
SDL_SCANCODE_KP_9 = 97,
SDL_SCANCODE_KP_0 = 98,
SDL_SCANCODE_KP_PERIOD = 99,

SDL_SCANCODE_NONUSBACKSLASH = 100, /**< This is the additional key
    that ISO
                                 *   keyboards have over ANSI ones,
                                 *   located between left shift and Y.
```

```
                                    *   Produces GRAVE ACCENT and TILDE
                                        in a
                                    *   US or UK Mac layout, REVERSE
                                        SOLIDUS
                                    *   (backslash) and VERTICAL LINE in
                                        a
                                    *   US or UK Windows layout, and
                                    *   LESS-THAN SIGN and GREATER-THAN
                                        SIGN
                                    *   in a Swiss German, German, or
                                        French
                                    *   layout. */
    SDL_SCANCODE_APPLICATION = 101, /**< windows contextual menu,
        compose */
    SDL_SCANCODE_POWER = 102, /**< The USB document says this is a
        status flag,
                            *   not a physical key - but some Mac
                                keyboards
                            *   do have a power key. */
    SDL_SCANCODE_KP_EQUALS = 103,
    SDL_SCANCODE_F13 = 104,
    SDL_SCANCODE_F14 = 105,
    SDL_SCANCODE_F15 = 106,
    SDL_SCANCODE_F16 = 107,
    SDL_SCANCODE_F17 = 108,
    SDL_SCANCODE_F18 = 109,
    SDL_SCANCODE_F19 = 110,
    SDL_SCANCODE_F20 = 111,
    SDL_SCANCODE_F21 = 112,
    SDL_SCANCODE_F22 = 113,
    SDL_SCANCODE_F23 = 114,
    SDL_SCANCODE_F24 = 115,
    SDL_SCANCODE_EXECUTE = 116,
    SDL_SCANCODE_HELP = 117, /**< AL Integrated Help Center */
    SDL_SCANCODE_MENU = 118, /**< Menu (show menu) */
    SDL_SCANCODE_SELECT = 119,
    SDL_SCANCODE_STOP = 120, /**< AC Stop */
    SDL_SCANCODE_AGAIN = 121, /**< AC Redo/Repeat */
    SDL_SCANCODE_UNDO = 122, /**< AC Undo */
    SDL_SCANCODE_CUT = 123,  /**< AC Cut */
    SDL_SCANCODE_COPY = 124, /**< AC Copy */
    SDL_SCANCODE_PASTE = 125, /**< AC Paste */
    SDL_SCANCODE_FIND = 126, /**< AC Find */
    SDL_SCANCODE_MUTE = 127,
    SDL_SCANCODE_VOLUMEUP = 128,
    SDL_SCANCODE_VOLUMEDOWN = 129,
/* not sure whether there's a reason to enable these */
/*    SDL_SCANCODE_LOCKINGCAPSLOCK = 130, */
/*    SDL_SCANCODE_LOCKINGNUMLOCK = 131, */
/*    SDL_SCANCODE_LOCKINGSCROLLLOCK = 132, */
```

```
SDL_SCANCODE_KP_COMMA = 133,
SDL_SCANCODE_KP_EQUALSAS400 = 134,

SDL_SCANCODE_INTERNATIONAL1 = 135, /**< used on Asian keyboards, see
                                     footnotes in USB doc */
SDL_SCANCODE_INTERNATIONAL2 = 136,
SDL_SCANCODE_INTERNATIONAL3 = 137, /**< Yen */
SDL_SCANCODE_INTERNATIONAL4 = 138,
SDL_SCANCODE_INTERNATIONAL5 = 139,
SDL_SCANCODE_INTERNATIONAL6 = 140,
SDL_SCANCODE_INTERNATIONAL7 = 141,
SDL_SCANCODE_INTERNATIONAL8 = 142,
SDL_SCANCODE_INTERNATIONAL9 = 143,
SDL_SCANCODE_LANG1 = 144, /**< Hangul/English toggle */
SDL_SCANCODE_LANG2 = 145, /**< Hanja conversion */
SDL_SCANCODE_LANG3 = 146, /**< Katakana */
SDL_SCANCODE_LANG4 = 147, /**< Hiragana */
SDL_SCANCODE_LANG5 = 148, /**< Zenkaku/Hankaku */
SDL_SCANCODE_LANG6 = 149, /**< reserved */
SDL_SCANCODE_LANG7 = 150, /**< reserved */
SDL_SCANCODE_LANG8 = 151, /**< reserved */
SDL_SCANCODE_LANG9 = 152, /**< reserved */

SDL_SCANCODE_ALTERASE = 153, /**< Erase-Eaze */
SDL_SCANCODE_SYSREQ = 154,
SDL_SCANCODE_CANCEL = 155,   /**< AC Cancel */
SDL_SCANCODE_CLEAR = 156,
SDL_SCANCODE_PRIOR = 157,
SDL_SCANCODE_RETURN2 = 158,
SDL_SCANCODE_SEPARATOR = 159,
SDL_SCANCODE_OUT = 160,
SDL_SCANCODE_OPER = 161,
SDL_SCANCODE_CLEARAGAIN = 162,
SDL_SCANCODE_CRSEL = 163,
SDL_SCANCODE_EXSEL = 164,

SDL_SCANCODE_KP_00 = 176,
SDL_SCANCODE_KP_000 = 177,
SDL_SCANCODE_THOUSANDSSEPARATOR = 178,
SDL_SCANCODE_DECIMALSEPARATOR = 179,
SDL_SCANCODE_CURRENCYUNIT = 180,
SDL_SCANCODE_CURRENCYSUBUNIT = 181,
SDL_SCANCODE_KP_LEFTPAREN = 182,
SDL_SCANCODE_KP_RIGHTPAREN = 183,
SDL_SCANCODE_KP_LEFTBRACE = 184,
SDL_SCANCODE_KP_RIGHTBRACE = 185,
SDL_SCANCODE_KP_TAB = 186,
SDL_SCANCODE_KP_BACKSPACE = 187,
SDL_SCANCODE_KP_A = 188,
SDL_SCANCODE_KP_B = 189,
```

```
SDL_SCANCODE_KP_C = 190,
SDL_SCANCODE_KP_D = 191,
SDL_SCANCODE_KP_E = 192,
SDL_SCANCODE_KP_F = 193,
SDL_SCANCODE_KP_XOR = 194,
SDL_SCANCODE_KP_POWER = 195,
SDL_SCANCODE_KP_PERCENT = 196,
SDL_SCANCODE_KP_LESS = 197,
SDL_SCANCODE_KP_GREATER = 198,
SDL_SCANCODE_KP_AMPERSAND = 199,
SDL_SCANCODE_KP_DBLAMPERSAND = 200,
SDL_SCANCODE_KP_VERTICALBAR = 201,
SDL_SCANCODE_KP_DBLVERTICALBAR = 202,
SDL_SCANCODE_KP_COLON = 203,
SDL_SCANCODE_KP_HASH = 204,
SDL_SCANCODE_KP_SPACE = 205,
SDL_SCANCODE_KP_AT = 206,
SDL_SCANCODE_KP_EXCLAM = 207,
SDL_SCANCODE_KP_MEMSTORE = 208,
SDL_SCANCODE_KP_MEMRECALL = 209,
SDL_SCANCODE_KP_MEMCLEAR = 210,
SDL_SCANCODE_KP_MEMADD = 211,
SDL_SCANCODE_KP_MEMSUBTRACT = 212,
SDL_SCANCODE_KP_MEMMULTIPLY = 213,
SDL_SCANCODE_KP_MEMDIVIDE = 214,
SDL_SCANCODE_KP_PLUSMINUS = 215,
SDL_SCANCODE_KP_CLEAR = 216,
SDL_SCANCODE_KP_CLEARENTRY = 217,
SDL_SCANCODE_KP_BINARY = 218,
SDL_SCANCODE_KP_OCTAL = 219,
SDL_SCANCODE_KP_DECIMAL = 220,
SDL_SCANCODE_KP_HEXADECIMAL = 221,

SDL_SCANCODE_LCTRL = 224,
SDL_SCANCODE_LSHIFT = 225,
SDL_SCANCODE_LALT = 226, /**< alt, option */
SDL_SCANCODE_LGUI = 227, /**< windows, command (apple), meta */
SDL_SCANCODE_RCTRL = 228,
SDL_SCANCODE_RSHIFT = 229,
SDL_SCANCODE_RALT = 230, /**< alt gr, option */
SDL_SCANCODE_RGUI = 231, /**< windows, command (apple), meta */

SDL_SCANCODE_MODE = 257, /**< I'm not sure if this is really not
    covered
                         *  by any of the above, but since there's
                             a
                         *  special SDL_KMOD_MODE for it I'm
                             adding it here
                         */
```

```
/* @} */// Usage page 0x07 */

/**
 *  \name Usage page 0x0C
 *
 *  These values are mapped from usage page 0x0C (USB consumer page).
 *  See https://usb.org/sites/default/files/hut1_2.pdf
 *
 *  There are way more keys in the spec than we can represent in the
 *  current scancode range, so pick the ones that commonly come up in
 *  real world usage.
 */
/* @{ */

SDL_SCANCODE_AUDIONEXT = 258,
SDL_SCANCODE_AUDIOPREV = 259,
SDL_SCANCODE_AUDIOSTOP = 260,
SDL_SCANCODE_AUDIOPLAY = 261,
SDL_SCANCODE_AUDIOMUTE = 262,
SDL_SCANCODE_MEDIASELECT = 263,
SDL_SCANCODE_WWW = 264,         /**< AL Internet Browser */
SDL_SCANCODE_MAIL = 265,
SDL_SCANCODE_CALCULATOR = 266,  /**< AL Calculator */
SDL_SCANCODE_COMPUTER = 267,
SDL_SCANCODE_AC_SEARCH = 268,   /**< AC Search */
SDL_SCANCODE_AC_HOME = 269,     /**< AC Home */
SDL_SCANCODE_AC_BACK = 270,     /**< AC Back */
SDL_SCANCODE_AC_FORWARD = 271,  /**< AC Forward */
SDL_SCANCODE_AC_STOP = 272,     /**< AC Stop */
SDL_SCANCODE_AC_REFRESH = 273,  /**< AC Refresh */
SDL_SCANCODE_AC_BOOKMARKS = 274, /**< AC Bookmarks */

/* @} */// Usage page 0x0C */

/**
 *  \name Walther keys
 *
 *  These are values that Christian Walther added (for mac
     keyboard?).
 */
/* @{ */

SDL_SCANCODE_BRIGHTNESSDOWN = 275,
SDL_SCANCODE_BRIGHTNESSUP = 276,
SDL_SCANCODE_DISPLAYSWITCH = 277, /**< display mirroring/dual display
                                     switch, video mode switch */
SDL_SCANCODE_KBDILLUMTOGGLE = 278,
SDL_SCANCODE_KBDILLUMDOWN = 279,
SDL_SCANCODE_KBDILLUMUP = 280,
SDL_SCANCODE_EJECT = 281,
```

```
SDL_SCANCODE_SLEEP = 282,       /**< SC System Sleep */

SDL_SCANCODE_APP1 = 283,
SDL_SCANCODE_APP2 = 284,

/* @} *//* Walther keys */

/**
 *  \name Usage page 0x0C (additional media keys)
 *
 *  These values are mapped from usage page 0x0C (USB consumer page).
 */
/* @{ */

SDL_SCANCODE_AUDIOREWIND = 285,
SDL_SCANCODE_AUDIOFASTFORWARD = 286,

/* @} *//* Usage page 0x0C (additional media keys) */

/**
 *  \name Mobile keys
 *
 *  These are values that are often used on mobile phones.
 */
/* @{ */

SDL_SCANCODE_SOFTLEFT = 287, /**< Usually situated below the display
    on phones and
                                used as a multi-function feature key
                                    for selecting
                                a software defined function shown on
                                    the bottom left
                                of the display. */
SDL_SCANCODE_SOFTRIGHT = 288, /**< Usually situated below the
    display on phones and
                                 used as a multi-function feature key
                                     for selecting
                                 a software defined function shown on
                                     the bottom right
                                 of the display. */
SDL_SCANCODE_CALL = 289, /**< Used for accepting phone calls. */
SDL_SCANCODE_ENDCALL = 290, /**< Used for rejecting phone calls. */

/* @} *//* Mobile keys */

/* Add any other keys here. */

SDL_NUM_SCANCODES = 512 /**< not a key, just marks the number of
    scancodes
                           for array bounds */
```

```
} SDL_Scancode;
```

Remarks

An SDL scancode is the physical representation of a key on the keyboard, independent of language and keyboard mapping.

Values of this type are used to represent keyboard keys, among other places in the `keysym.scancode` field of the SDL_KeyboardEvent structure.

The values in this enumeration are based on the USB usage page standard: https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf

Version

This enum is available since SDL 3.0.0.

SDL_Semaphore

A means to manage access to a resource, by count, between threads.

Header File

Defined in SDL3/SDL_mutex.h

Syntax

```
typedef struct SDL_Semaphore SDL_Semaphore;
```

Remarks

Semaphores (specifically, "counting semaphores"), let X number of threads request access at the same time, each thread granted access decrementing a counter. When the counter reaches zero, future requests block until a prior thread releases their request, incrementing the counter again.

Wikipedia has a thorough explanation of the concept:
https://en.wikipedia.org/wiki/Semaphore_(programming)

Version

This struct is available since SDL 3.0.0.

SDL_SensorEvent

Sensor event structure (event.sensor.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_SensorEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_SENSOR_UPDATE */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_SensorID which; /**< The instance ID of the sensor */
    float data[6];    /**< Up to 6 values from the sensor - additional
        values can be queried using SDL_GetSensorData() */
    Uint64 sensor_timestamp; /**< The timestamp of the sensor reading in
        nanoseconds, not necessarily synchronized with the system clock
        */
} SDL_SensorEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_SensorID

This is a unique ID for a sensor for the time it is connected to the system, and is never reused for the lifetime of the application.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
typedef Uint32 SDL_SensorID;
```

Remarks

The ID value starts at 1 and increments from there. The value 0 is an invalid ID.

Version

This datatype is available since SDL 3.0.0.

SDL_SensorType

The different sensors defined by SDL.

Header File

Defined in SDL3/SDL_sensor.h

Syntax

```
typedef enum SDL_SensorType
{
    SDL_SENSOR_INVALID = -1, /**< Returned for an invalid sensor */
    SDL_SENSOR_UNKNOWN,      /**< Unknown sensor type */
    SDL_SENSOR_ACCEL,        /**< Accelerometer */
    SDL_SENSOR_GYRO,         /**< Gyroscope */
    SDL_SENSOR_ACCEL_L,      /**< Accelerometer for left Joy-Con
        controller and Wii nunchuk */
    SDL_SENSOR_GYRO_L,       /**< Gyroscope for left Joy-Con controller
        */
    SDL_SENSOR_ACCEL_R,      /**< Accelerometer for right Joy-Con
        controller */
    SDL_SENSOR_GYRO_R        /**< Gyroscope for right Joy-Con controller
        */
} SDL_SensorType;
```

Remarks

Additional sensors may be available, using platform dependent semantics.

Here are the additional Android sensors:

https://developer.android.com/reference/android/hardware/SensorEvent.html#values

Accelerometer sensor notes:

The accelerometer returns the current acceleration in SI meters per second squared. This measurement includes the force of gravity, so a device at rest will have an value of SDL_STANDARD_GRAVITY away from the center of the earth, which is a positive Y value.

- `values[0]`: Acceleration on the x axis
- `values[1]`: Acceleration on the y axis
- `values[2]`: Acceleration on the z axis

For phones and tablets held in natural orientation and game controllers held in front of you, the axes are defined as follows:

- -X ... +X : left ... right

- -Y ... +Y : bottom ... top
- -Z ... +Z : farther ... closer

The accelerometer axis data is not changed when the device is rotated.

Gyroscope sensor notes:

The gyroscope returns the current rate of rotation in radians per second. The rotation is positive in the counter-clockwise direction. That is, an observer looking from a positive location on one of the axes would see positive rotation on that axis when it appeared to be rotating counter-clockwise.

- `values[0]`: Angular speed around the x axis (pitch)
- `values[1]`: Angular speed around the y axis (yaw)
- `values[2]`: Angular speed around the z axis (roll)

For phones and tablets held in natural orientation and game controllers held in front of you, the axes are defined as follows:

- -X ... +X : left ... right
- -Y ... +Y : bottom ... top
- -Z ... +Z : farther ... closer

The gyroscope axis data is not changed when the device is rotated.

Version

This enum is available since SDL 3.0.0.

See Also

- SDL_GetCurrentDisplayOrientation

SDL_SpinLock

An atomic spinlock.

Header File

Defined in SDL3/SDL_atomic.h

Syntax

```
typedef int SDL_SpinLock;
```

Remarks

The atomic locks are efficient spinlocks using CPU instructions, but are vulnerable to starvation and can spin forever if a thread holding a lock has been terminated. For this reason you should minimize the code executed inside an atomic lock and never do expensive things like API or system calls while holding them.

They are also vulnerable to starvation if the thread holding the lock is lower priority than other threads and doesn't get scheduled. In general you should use mutexes instead, since they have better performance and contention behavior.

The atomic locks are not safe to lock recursively.

Porting Note: The spin lock functions and type are required and can not be emulated because they are used in the atomic emulation code.

SDL_Storage

An abstract interface for filesystem access.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
typedef struct SDL_Storage SDL_Storage;
```

Remarks

This is an opaque datatype. One can create this object using standard SDL functions like SDL_OpenTitleStorage or SDL_OpenUserStorage, etc, or create an object with a custom implementation using SDL_OpenStorage.

Version

This struct is available since SDL 3.0.0.

SDL_StorageInterface

Function interface for SDL_Storage.

Header File

Defined in SDL3/SDL_storage.h

Syntax

```
typedef struct SDL_StorageInterface
{
    /* Called when the storage is closed */
    int (SDLCALL *close)(void *userdata);

    /* Optional, returns whether the storage is currently ready for
        access */
    SDL_bool (SDLCALL *ready)(void *userdata);

    /* Enumerate a directory, optional for write-only storage */
    int (SDLCALL *enumerate)(void *userdata, const char *path,
        SDL_EnumerateDirectoryCallback callback, void
        *callback_userdata);

    /* Get path information, optional for write-only storage */
    int (SDLCALL *info)(void *userdata, const char *path, SDL_PathInfo
        *info);

    /* Read a file from storage, optional for write-only storage */
    int (SDLCALL *read_file)(void *userdata, const char *path, void
        *destination, Uint64 length);

    /* Write a file to storage, optional for read-only storage */
    int (SDLCALL *write_file)(void *userdata, const char *path, const
        void *source, Uint64 length);

    /* Create a directory, optional for read-only storage */
    int (SDLCALL *mkdir)(void *userdata, const char *path);

    /* Remove a file or empty directory, optional for read-only storage
        */
    int (SDLCALL *remove)(void *userdata, const char *path);

    /* Rename a path, optional for read-only storage */
    int (SDLCALL *rename)(void *userdata, const char *oldpath, const
        char *newpath);

    /* Get the space remaining, optional for read-only storage */
```

```
    Uint64 (SDLCALL *space_remaining)(void *userdata);
} SDL_StorageInterface;
```

Remarks

Apps that want to supply a custom implementation of SDL_Storage will fill in all the functions in this struct, and then pass it to SDL_OpenStorage to create a custom SDL_Storage object.

It is not usually necessary to do this; SDL provides standard implementations for many things you might expect to do with an SDL_Storage.

Version

This struct is available since SDL 3.0.0.

SDL_Surface

A collection of pixels used in software blitting.

Header File

Defined in SDL3/SDL_surface.h

Syntax

```
typedef struct SDL_Surface
{
    Uint32 flags;            /**< Read-only */
    SDL_PixelFormat *format; /**< Read-only */
    int w, h;                /**< Read-only */
    int pitch;               /**< Read-only */
    void *pixels;            /**< Read-write */

    void *reserved;          /**< Private */

    /** information needed for surfaces requiring locks */
    int locked;              /**< Read-only */

    /** list of BlitMap that hold a reference to this surface */
    void *list_blitmap;      /**< Private */

    /** clipping information */
    SDL_Rect clip_rect;      /**< Read-only */

    /** info for fast blit mapping to other surfaces */
    SDL_BlitMap *map;        /**< Private */

    /** Reference count -- used when freeing surface */
    int refcount;            /**< Read-mostly */
} SDL_Surface;
```

Remarks

Pixels are arranged in memory in rows, with the top row first. Each row occupies an amount of memory given by the pitch (sometimes known as the row stride in non-SDL APIs).

Within each row, pixels are arranged from left to right until the width is reached. Each pixel occupies a number of bits appropriate for its format, with most formats representing each pixel as one or more whole bytes (in some indexed formats, instead multiple pixels are packed into each byte), and a byte order given by the format. After encoding all pixels, any remaining bytes to

reach the pitch are used as padding to reach a desired alignment, and have undefined contents.

Version

This struct is available since SDL 3.0.0.

SDL_SystemCursor

Cursor types for SDL_CreateSystemCursor().

Header File

Defined in SDL3/SDL_mouse.h

Syntax

```
typedef enum SDL_SystemCursor
{
    SDL_SYSTEM_CURSOR_ARROW,  /**< Arrow */
    SDL_SYSTEM_CURSOR_IBEAM,  /**< I-beam */
    SDL_SYSTEM_CURSOR_WAIT,   /**< Wait */
    SDL_SYSTEM_CURSOR_CROSSHAIR, /**< Crosshair */
    SDL_SYSTEM_CURSOR_WAITARROW, /**< Small wait cursor (or Wait if not
        available) */
    SDL_SYSTEM_CURSOR_SIZENWSE, /**< Double arrow pointing northwest and
        southeast */
    SDL_SYSTEM_CURSOR_SIZENESW, /**< Double arrow pointing northeast and
        southwest */
    SDL_SYSTEM_CURSOR_SIZEWE, /**< Double arrow pointing west and east */
    SDL_SYSTEM_CURSOR_SIZENS, /**< Double arrow pointing north and south
        */
    SDL_SYSTEM_CURSOR_SIZEALL, /**< Four pointed arrow pointing north,
        south, east, and west */
    SDL_SYSTEM_CURSOR_NO,     /**< Slashed circle or crossbones */
    SDL_SYSTEM_CURSOR_HAND,   /**< Hand */
    SDL_SYSTEM_CURSOR_WINDOW_TOPLEFT, /**< Window resize top-left (or
        SIZENWSE) */
    SDL_SYSTEM_CURSOR_WINDOW_TOP,     /**< Window resize top (or SIZENS)
        */
    SDL_SYSTEM_CURSOR_WINDOW_TOPRIGHT, /**< Window resize top-right (or
        SIZENESW) */
    SDL_SYSTEM_CURSOR_WINDOW_RIGHT,   /**< Window resize right (or
        SIZEWE) */
    SDL_SYSTEM_CURSOR_WINDOW_BOTTOMRIGHT, /**< Window resize
        bottom-right (or SIZENWSE) */
    SDL_SYSTEM_CURSOR_WINDOW_BOTTOM,  /**< Window resize bottom (or
        SIZENS) */
    SDL_SYSTEM_CURSOR_WINDOW_BOTTOMLEFT, /**< Window resize bottom-left
        (or SIZENESW) */
    SDL_SYSTEM_CURSOR_WINDOW_LEFT,    /**< Window resize left (or
        SIZEWE) */
    SDL_NUM_SYSTEM_CURSORS
} SDL_SystemCursor;
```

Version

This enum is available since SDL 3.0.0.

SDL_SystemTheme

System theme.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef enum SDL_SystemTheme
{
    SDL_SYSTEM_THEME_UNKNOWN, /**< Unknown system theme */
    SDL_SYSTEM_THEME_LIGHT,  /**< Light colored system theme */
    SDL_SYSTEM_THEME_DARK    /**< Dark colored system theme */
} SDL_SystemTheme;
```

Version

This enum is available since SDL 3.0.0.

SDL_TextEditingEvent

Keyboard text editing event structure (event.edit.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_TextEditingEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_TEXT_EDITING */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window with keyboard focus, if any */
    char *text;       /**< The editing text */
    Sint32 start;     /**< The start cursor of selected editing text */
    Sint32 length;    /**< The length of selected editing text */
} SDL_TextEditingEvent;
```

Remarks

The `text` is owned by SDL and should be copied if the application wants to hold onto it beyond the scope of handling this event.

Version

This struct is available since SDL 3.0.0.

SDL_TextInputEvent

Keyboard text input event structure (event.text.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_TextInputEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_TEXT_INPUT */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The window with keyboard focus, if any */
    char *text;       /**< The input text, UTF-8 encoded */
} SDL_TextInputEvent;
```

Remarks

The `text` is owned by SDL and should be copied if the application wants to hold onto it beyond the scope of handling this event.

Version

This struct is available since SDL 3.0.0.

SDL_Texture

An efficient driver-specific representation of pixel data

Header File

Defined in SDL3/SDL_render.h

Syntax

```
struct SDL_Texture;
```

Version

This struct is available since SDL 3.0.0.

SDL_TextureAccess

The access pattern allowed for a texture.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
typedef enum SDL_TextureAccess
{
    SDL_TEXTUREACCESS_STATIC, /**< Changes rarely, not lockable */
    SDL_TEXTUREACCESS_STREAMING, /**< Changes frequently, lockable */
    SDL_TEXTUREACCESS_TARGET  /**< Texture can be used as a render
        target */
} SDL_TextureAccess;
```

Version

This enum is available since SDL 3.0.0.

SDL_ThreadFunction

The function passed to SDL_CreateThread().

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
typedef int (SDLCALL * SDL_ThreadFunction) (void *data);
```

Function Parameters

data	what was passed as `data` to SDL_CreateThread()

Return Value

Returns a value that can be reported through SDL_WaitThread().

Version

This datatype is available since SDL 3.0.0.

SDL_ThreadPriority

The SDL thread priority.

Header File

Defined in SDL3/SDL_thread.h

Syntax

```
typedef enum SDL_ThreadPriority {
    SDL_THREAD_PRIORITY_LOW,
    SDL_THREAD_PRIORITY_NORMAL,
    SDL_THREAD_PRIORITY_HIGH,
    SDL_THREAD_PRIORITY_TIME_CRITICAL
} SDL_ThreadPriority;
```

Remarks

SDL will make system changes as necessary in order to apply the thread priority. Code which attempts to control thread state related to priority should be aware that calling SDL_SetThreadPriority may alter such state. SDL_HINT_THREAD_PRIORITY_POLICY can be used to control aspects of this behavior.

Version

This enum is available since SDL 3.0.0.

SDL_TimeFormat

The preferred time format of the current system locale.

Header File

Defined in SDL3/SDL_time.h

Syntax

```
typedef enum SDL_TimeFormat
{
    SDL_TIME_FORMAT_24HR = 0, /**< 24 hour time */
    SDL_TIME_FORMAT_12HR = 1 /**< 12 hour time */
} SDL_TimeFormat;
```

Version

This enum is available since SDL 3.0.0.

See Also

- SDL_PROP_GLOBAL_SYSTEM_TIME_FORMAT_NUMBER

SDL_TimerCallback

Function prototype for the timer callback function.

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
typedef Uint32 (SDLCALL *SDL_TimerCallback)(Uint32 interval, void
    *param);
```

Function Parameters

interval	the current callback time interval.
param	an arbitrary pointer provided by the app through SDL_AddTimer, for its own use.

Return Value

Returns the new callback time interval, or 0 to disable further runs of the callback.

Remarks

The callback function is passed the current timer interval and returns the next timer interval, in milliseconds. If the returned value is the same as the one passed in, the periodic alarm continues, otherwise a new alarm is scheduled. If the callback returns 0, the periodic alarm is cancelled.

Thread Safety

SDL may call this callback at any time from a background thread; the application is responsible for locking resources the callback touches that need to be protected.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_AddTimer

SDL_TimerID

Definition of the timer ID type.

Header File

Defined in SDL3/SDL_timer.h

Syntax

```
typedef Uint32 SDL_TimerID;
```

Version

This datatype is available since SDL 3.0.0.

SDL_TouchFingerEvent

Touch finger event structure (event.tfinger.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_TouchFingerEvent
{
    SDL_EventType type; /**< ::SDL_EVENT_FINGER_MOTION or
        ::SDL_EVENT_FINGER_DOWN or ::SDL_EVENT_FINGER_UP */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_TouchID touchID; /**< The touch device id */
    SDL_FingerID fingerID;
    float x;          /**< Normalized in the range 0...1 */
    float y;          /**< Normalized in the range 0...1 */
    float dx;         /**< Normalized in the range -1...1 */
    float dy;         /**< Normalized in the range -1...1 */
    float pressure;   /**< Normalized in the range 0...1 */
    SDL_WindowID windowID; /**< The window underneath the finger, if any
        */
} SDL_TouchFingerEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_TransferCharacteristics

The color transfer characteristics.

Header File

Defined in SDL3/SDL_pixels.h

Syntax

```
typedef enum SDL_TransferCharacteristics
{
    SDL_TRANSFER_CHARACTERISTICS_UNKNOWN = 0,
    SDL_TRANSFER_CHARACTERISTICS_BT709 = 1,    /**< Rec. ITU-R BT.709-6
        / ITU-R BT1361 */
    SDL_TRANSFER_CHARACTERISTICS_UNSPECIFIED = 2,
    SDL_TRANSFER_CHARACTERISTICS_GAMMA22 = 4,  /**< ITU-R BT.470-6
        System M / ITU-R BT1700 625 PAL & SECAM */
    SDL_TRANSFER_CHARACTERISTICS_GAMMA28 = 5,  /**< ITU-R BT.470-6
        System B, G */
    SDL_TRANSFER_CHARACTERISTICS_BT601 = 6,    /**< SMPTE ST 170M /
        ITU-R BT.601-7 525 or 625 */
    SDL_TRANSFER_CHARACTERISTICS_SMPTE240 = 7, /**< SMPTE ST 240M */
    SDL_TRANSFER_CHARACTERISTICS_LINEAR = 8,
    SDL_TRANSFER_CHARACTERISTICS_LOG100 = 9,
    SDL_TRANSFER_CHARACTERISTICS_LOG100_SQRT10 = 10,
    SDL_TRANSFER_CHARACTERISTICS_IEC61966 = 11, /**< IEC 61966-2-4 */
    SDL_TRANSFER_CHARACTERISTICS_BT1361 = 12,  /**< ITU-R BT1361
        Extended Colour Gamut */
    SDL_TRANSFER_CHARACTERISTICS_SRGB = 13,    /**< IEC 61966-2-1 (sRGB
        or sYCC) */
    SDL_TRANSFER_CHARACTERISTICS_BT2020_10BIT = 14, /**< ITU-R BT2020
        for 10-bit system */
    SDL_TRANSFER_CHARACTERISTICS_BT2020_12BIT = 15, /**< ITU-R BT2020
        for 12-bit system */
    SDL_TRANSFER_CHARACTERISTICS_PQ = 16,      /**< SMPTE ST 2084 for
        10-, 12-, 14- and 16-bit systems */
    SDL_TRANSFER_CHARACTERISTICS_SMPTE428 = 17, /**< SMPTE ST 428-1 */
    SDL_TRANSFER_CHARACTERISTICS_HLG = 18,     /**< ARIB STD-B67, known
        as "hybrid log-gamma" (HLG) */
    SDL_TRANSFER_CHARACTERISTICS_CUSTOM = 31
} SDL_TransferCharacteristics;
```

Remarks

These are as described by https://www.itu.int/rec/T-REC-H.273-201612-S/en

Version

This enum is available since SDL 3.0.0.

SDL_UserEvent

A user-defined event type (event.user.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_UserEvent
{
    Uint32 type;      /**< ::SDL_EVENT_USER through ::SDL_EVENT_LAST-1,
        Uint32 because these are not in the SDL_EventType enumeration */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The associated window if any */
    Sint32 code;      /**< User defined event code */
    void *data1;      /**< User defined data pointer */
    void *data2;      /**< User defined data pointer */
} SDL_UserEvent;
```

Remarks

This event is unique; it is never created by SDL, but only by the application. The event can be pushed onto the event queue using SDL_PushEvent(). The contents of the structure members are completely up to the programmer; the only requirement is that ”’type”’ is a value obtained from SDL_RegisterEvents().

Code Examples

```
extern Sint32 my_event_code;
extern void *significant_data;
extern void *some_other_data;

const Uint32 myEventType = SDL_RegisterEvents(1);
if (myEventType != ((Uint32)-1)) {
    SDL_Event event;
    SDL_zero(event);
    event.type = myEventType;
    event.user.code = my_event_code;
    event.user.data1 = significant_data;
    event.user.data2 = some_other_data;
    SDL_PushEvent(&event);
}
```

Version

This struct is available since SDL 3.0.0.

SDL_Version

Information about the version of SDL in use.

Header File

Defined in SDL3/SDL_version.h

Syntax

```
typedef struct SDL_Version
{
    Uint8 major;      /**< major version */
    Uint8 minor;      /**< minor version */
    Uint8 patch;      /**< update version */
} SDL_Version;
```

Remarks

Represents the library's version as three levels: major revision (increments with massive changes, additions, and enhancements), minor revision (increments with backwards-compatible changes to the major revision), and patchlevel (increments with fixes to the minor revision).

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_VERSION
- SDL_GetVersion

SDL_Vertex

Vertex structure.

Header File

Defined in SDL3/SDL_render.h

Syntax

```
typedef struct SDL_Vertex
{
    SDL_FPoint position;     /**< Vertex position, in SDL_Renderer
        coordinates */
    SDL_FColor color;        /**< Vertex color */
    SDL_FPoint tex_coord;    /**< Normalized texture coordinates, if
        needed */
} SDL_Vertex;
```

Version

This struct is available since SDL 3.0.0.

SDL_VirtualJoystickDesc

The structure that defines an extended virtual joystick description

Header File

Defined in SDL3/SDL_joystick.h

Syntax

```
typedef struct SDL_VirtualJoystickDesc
{
    Uint16 type;      /**< `SDL_JoystickType` */
    Uint16 naxes;     /**< the number of axes on this joystick */
    Uint16 nbuttons;  /**< the number of buttons on this joystick */
    Uint16 nhats;     /**< the number of hats on this joystick */
    Uint16 vendor_id; /**< the USB vendor ID of this joystick */
    Uint16 product_id; /**< the USB product ID of this joystick */
    Uint16 padding;   /**< unused */
    Uint32 button_mask; /**< A mask of which buttons are valid for this
        controller
                         e.g. (1 << SDL_GAMEPAD_BUTTON_SOUTH) */
    Uint32 axis_mask; /**< A mask of which axes are valid for this
        controller
                         e.g. (1 << SDL_GAMEPAD_AXIS_LEFTX) */
    const char *name; /**< the name of the joystick */

    void *userdata;   /**< User data pointer passed to callbacks */
    void (SDLCALL *Update)(void *userdata); /**< Called when the
        joystick state should be updated */
    void (SDLCALL *SetPlayerIndex)(void *userdata, int player_index);
        /**< Called when the player index is set */
    int (SDLCALL *Rumble)(void *userdata, Uint16 low_frequency_rumble,
        Uint16 high_frequency_rumble); /**< Implements
        SDL_RumbleJoystick() */
    int (SDLCALL *RumbleTriggers)(void *userdata, Uint16 left_rumble,
        Uint16 right_rumble); /**< Implements
        SDL_RumbleJoystickTriggers() */
    int (SDLCALL *SetLED)(void *userdata, Uint8 red, Uint8 green, Uint8
        blue); /**< Implements SDL_SetJoystickLED() */
    int (SDLCALL *SendEffect)(void *userdata, const void *data, int
        size); /**< Implements SDL_SendJoystickEffect() */
} SDL_VirtualJoystickDesc;
```

Remarks

All elements of this structure are optional and can be left 0.

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_AttachVirtualJoystickEx

SDL_Window

The struct used as an opaque handle to a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef struct SDL_Window SDL_Window;
```

Version

This struct is available since SDL 3.0.0.

See Also

- SDL_CreateWindow

SDL_WindowEvent

Window state change event data (event.window.*)

Header File

Defined in SDL3/SDL_events.h

Syntax

```
typedef struct SDL_WindowEvent
{
    SDL_EventType type; /**< ::SDL_WINDOWEVENT_* */
    Uint32 reserved;
    Uint64 timestamp; /**< In nanoseconds, populated using
        SDL_GetTicksNS() */
    SDL_WindowID windowID; /**< The associated window */
    Sint32 data1;     /**< event dependent data */
    Sint32 data2;     /**< event dependent data */
} SDL_WindowEvent;
```

Version

This struct is available since SDL 3.0.0.

SDL_WindowFlags

The flags on a window.

Header File

Defined in SDL3/SDL_video.h

Syntax

```
typedef Uint32 SDL_WindowFlags;

#define SDL_WINDOW_FULLSCREEN       0x00000001U /**< window is in
    fullscreen mode */
#define SDL_WINDOW_OPENGL           0x00000002U /**< window usable with
    OpenGL context */
#define SDL_WINDOW_OCCLUDED         0x00000004U /**< window is occluded
    */
#define SDL_WINDOW_HIDDEN           0x00000008U /**< window is neither
    mapped onto the desktop nor shown in the taskbar/dock/window list;
    SDL_ShowWindow() is required for it to become visible */
#define SDL_WINDOW_BORDERLESS       0x00000010U /**< no window
    decoration */
#define SDL_WINDOW_RESIZABLE        0x00000020U /**< window can be
    resized */
#define SDL_WINDOW_MINIMIZED        0x00000040U /**< window is minimized
    */
#define SDL_WINDOW_MAXIMIZED        0x00000080U /**< window is maximized
    */
#define SDL_WINDOW_MOUSE_GRABBED    0x00000100U /**< window has grabbed
    mouse input */
#define SDL_WINDOW_INPUT_FOCUS      0x00000200U /**< window has input
    focus */
#define SDL_WINDOW_MOUSE_FOCUS      0x00000400U /**< window has mouse
    focus */
#define SDL_WINDOW_EXTERNAL         0x00000800U /**< window not created
    by SDL */
#define SDL_WINDOW_MODAL            0x00001000U /**< window is modal */
#define SDL_WINDOW_HIGH_PIXEL_DENSITY 0x00002000U /**< window uses high
    pixel density back buffer if possible */
#define SDL_WINDOW_MOUSE_CAPTURE    0x00004000U /**< window has mouse
    captured (unrelated to MOUSE_GRABBED) */
#define SDL_WINDOW_ALWAYS_ON_TOP    0x00008000U /**< window should
    always be above others */
#define SDL_WINDOW_UTILITY          0x00020000U /**< window should be
    treated as a utility window, not showing in the task bar and window
    list */
```

```
#define SDL_WINDOW_TOOLTIP          0x00040000U /**< window should be
    treated as a tooltip and does not get mouse or keyboard focus,
    requires a parent window */
#define SDL_WINDOW_POPUP_MENU       0x00080000U /**< window should be
    treated as a popup menu, requires a parent window */
#define SDL_WINDOW_KEYBOARD_GRABBED 0x00100000U /**< window has grabbed
    keyboard input */
#define SDL_WINDOW_VULKAN           0x10000000U /**< window usable for
    Vulkan surface */
#define SDL_WINDOW_METAL            0x20000000U /**< window usable for
    Metal view */
#define SDL_WINDOW_TRANSPARENT      0x40000000U /**< window with
    transparent buffer */
#define SDL_WINDOW_NOT_FOCUSABLE    0x80000000U /**< window should not
    be focusable */
```

Remarks

These cover a lot of true/false, or on/off, window state. Some of it is immutable after being set through SDL_CreateWindow(), some of it can be changed on existing windows by the app, and some of it might be altered by the user or system outside of the app's control.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_GetWindowFlags

SDL_WindowsMessageHook

A callback to be used with SDL_SetWindowsMessageHook.

Header File

Defined in SDL3/SDL_system.h

Syntax

```
typedef SDL_bool (SDLCALL *SDL_WindowsMessageHook)(void *userdata, MSG
    *msg);
```

Function Parameters

userdata	the app-defined pointer provided to SDL_SetWindowsMessageHook.
msg	a pointer to a Win32 event structure to process.

Return Value

Returns SDL_TRUE to let event continue on, SDL_FALSE to drop it.

Remarks

This callback may modify the message, and should return SDL_TRUE if the message should continue to be processed, or SDL_FALSE to prevent further processing.

As this is processing a message directly from the Windows event loop, this callback should do the minimum required work and return quickly.

Code Examples

```
#if defined(SDL_PLATFORM_WIN32) || defined(SDL_PLATFORM_GDK)
SDL_bool MyMessageHook(void *userdata, MSG *msg)
{
    // do things with userdata and msg...

    return SDL_TRUE; // let SDL continue processing the message
}

// ...
SDL_SetWindowsMessageHook(MyMessageHook, NULL);
#endif
```

Thread Safety

This may only be called (by SDL) from the thread handling the Windows event loop.

Version

This datatype is available since SDL 3.0.0.

See Also

- SDL_SetWindowsMessageHook
- SDL_HINT_WINDOWS_ENABLE_MESSAGELOOP

SDL_WinRT_DeviceFamily

WinRT Device Family

Header File

Defined in SDL3/SDL_system.h

Syntax

```
typedef enum SDL_WinRT_DeviceFamily
{
    /** Unknown family */
    SDL_WINRT_DEVICEFAMILY_UNKNOWN,

    /** Desktop family*/
    SDL_WINRT_DEVICEFAMILY_DESKTOP,

    /** Mobile family (for example smartphone) */
    SDL_WINRT_DEVICEFAMILY_MOBILE,

    /** XBox family */
    SDL_WINRT_DEVICEFAMILY_XBOX,
} SDL_WinRT_DeviceFamily;
```

Version

This enum is available since SDL 3.0.0.

SDL_WinRT_Path

WinRT / Windows Phone path types

Header File

Defined in SDL3/SDL_system.h

Syntax

```
typedef enum SDL_WinRT_Path
{
    /** The installed app's root directory.
        Files here are likely to be read-only. */
    SDL_WINRT_PATH_INSTALLED_LOCATION,

    /** The app's local data store. Files may be written here */
    SDL_WINRT_PATH_LOCAL_FOLDER,

    /** The app's roaming data store. Unsupported on Windows Phone.
        Files written here may be copied to other machines via a network
        connection.
    */
    SDL_WINRT_PATH_ROAMING_FOLDER,

    /** The app's temporary data store. Unsupported on Windows Phone.
        Files written here may be deleted at any time. */
    SDL_WINRT_PATH_TEMP_FOLDER
} SDL_WinRT_Path;
```

Version

This enum is available since SDL 3.0.0.