The Tobii Stream Engine API consists of the following modules.

- tobii - Core functions.
- tobii_streams - Basic gaze- and data streams.
- tobii_wearable - Gaze data streams for wearable/vr devices.
- tobii_engine - Communication with the Tobii Engine service

The tobii.h header file collects the core API functions of stream engine. It contains functions to initialize the API and establish a connection to a tracker, as well as enumerating connected devices and requesting callbacks for subscriptions. There are also functions for querying the current state of a tracker, and to query its capabilities.

The API documentation includes example code snippets that shows the use of each function, they don't necessarily describe the best practice in which to use the api. For a more in-depth example of the best practices, see the samples that are supplied together with the stream engine library.

Thread safety

The Tobii Stream Engine API implements full thread safety across all API functions. However, it is up to the user to guarantee thread safety in code injected into Stream Engine, for example inside callbacks or if a custom memory allocator is supplied. It is not allowed to call Stream Engine API functions from within a callback invoked by stream engine. Attempting to do so will result in TOBII_ERROR_CALLBACK_IN_PROGRESS. A specific exception to this is tobii_system_clock() which specifically *is* allowed to be called even from within a callback function.

In the *samples* folder, you can find complete examples on how to use Stream Engine with multiple threads, such as *background_thread_sample* and *game_loop_sample*.

tobii_error_message

Function

Returns a printable error message.

Syntax

```
#include <tobii/tobii.h>
char const* tobii_error_message( tobii_error_t error );
```

Remarks

All other functions in the API returns an error code from the tobii_error_t enumeration. tobii_error_message translates from these error codes to a human readable message. If the value passed in the *error* parameter is not within the range of the tobii_error_t enum, a generic message is returned.

Return value

tobii_error_message returns a zero-terminated C string describing the specified error code. The string returned is statically allocated, so it should not be freed.

Example

```
#include <tobii/tobii.h>
#include <stdio.h>

int main()
{
    tobii_api_t* api;

    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    if( error != TOBII_ERROR_NO_ERROR ) printf( "%s\n", tobii_error_message( error ) );

    error = tobii_api_destroy( api );
    if( error != TOBII_ERROR_NO_ERROR ) printf( "%s\n", tobii_error_message( error ) );

    return 0;
}
```

tobii_get_api_version

Function

Query the current version of the API.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_get_api_version( tobii_version_t* version );
```

Remarks

tobii_get_api_version can be used to query the version of the stream engine dll currently used.

version is a pointer to an tobii_version_t variable to receive the current version numbers. It

contains the following members:

- *major* incremented for API changes which are not backward-compatible.
- *minor* incremented for releases which add new, but backward-compatible, API features.
- *revision* incremented for minor changes and bug fixes which do not change the API.
- *build* incremented every time a new build is done, even when there are no changes.

Return value

If the call is successful, tobii_get_api_version returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_get_api_version returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *version* parameter was passed in as NULL. *version* is not optional.

Example

```
#include <tobii/tobii.h>
#include <stdio.h>

int main()
{
    tobii_version_t version;
    tobii_error_t error = tobii_get_api_version( &version );
    if( error == TOBII_ERROR_NO_ERROR )
        printf( "Current API version: %d.%d.%d\n", version.major, version.minor,
            version.revision );

    return 0;
}
```

tobii_api_create

Function

Initializes the stream engine API, with optionally provided custom memory allocation and logging functions.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_api_create( tobii_api_t** api,
    tobii_custom_alloc_t const* custom_alloc, tobii_custom_log_t const* custom_log );
```

Remarks

Before any other API function can be invoked (with the exception of tobii_error_message and tobii_get_api_version), the API needs to be set up for use, by calling tobii_api_create. The resulting tobii_api_t instance is passed explicitly to some functions, or implicitly to some by passing a device instance. When creating an API instance, it is possible, but not necessary, to customize the behavior by passing one or more of the optional parameters *custom_alloc* and *custom_log*.

api must be a pointer to a variable of the type tobii_api_t* that is, a pointer to a tobii_api_t-pointer. This variable will be filled in with a pointer to the created instance. tobii_api_t is an opaque type, and only its declaration is available in the API.

custom_alloc is used to specify a custom allocator for dynamic memory. A custom allocator is specified as a pointer to a tobii_custom_alloc_t instance, which has the following fields:

- *mem_context* a custom user data pointer which will be passed through unmodified to the allocator functions when they are called.
- *malloc_func* a pointer to a function implementing allocation of memory. It must have the following signature:

  ```
  void* custom_malloc( void* mem_context, size_t size )
  ```

  where *mem_context* will be the same value as the *mem_context* field of tobii_custom_alloc_t, and *size* is the number of bytes to allocate. The function must return a pointer to a memory area of, at least, *size* bytes, but may return NULL if memory could not be allocated, in which case the API function invoking the allocation will fail and return the error **TOBII_ERROR_ALLOCATION_FAILED**.
- *free_func* a pointer to a function implementing deallocation of memory. It must have the following signature:

  ```
  void custom_free( void* mem_context, void* ptr )
  ```

where *mem_context* will be the same value as the *mem_context* field of tobii_custom_alloc_t, and *ptr* is a pointer to the memory block (as returned by a call to the custom malloc_func) to be released. The value of *ptr* will never be NULL, and only a single call to free_func will be made for each call made to malloc_func.

custom_alloc is an optional parameter, and may be NULL, in which case a default allocator is used.

NOTE: Stream engine does not guarantee thread safety on *custom_alloc*. If thread safety is a requirement, it should be satisfied in the implementation of *custom_alloc*. Default allocator runs thread safe.

custom_log is used to specify a custom function to handle log printouts. A custom logger is specified as a pointer to a tobii_custom_log_t instance, which has the following fields:

- *log_context* a custom user data pointer which will be passed through unmodified to the custom log function when it is called.
- *log_func* a pointer to a function implementing allocation of memory. It must have the following signature:

  ```
  void custom_log( void* log_context, tobii_log_level_t level, char const* text )
  ```

  where *log_context* will be the same value as the *log_context* field of tobii_custom_log_t, *level* is one of the log levels defined in the tobii_log_level_t enum:

  - **TOBII_LOG_LEVEL_ERROR**
  - **TOBII_LOG_LEVEL_WARN**
  - **TOBII_LOG_LEVEL_INFO**
  - **TOBII_LOG_LEVEL_DEBUG**
  - **TOBII_LOG_LEVEL_TRACE**

  and *text* is the message to be logged. The *level* parameter can be used for filtering log messages by severity, but it is up to the custom log function how to make use of it.

custom_log is an optional parameter, and may be NULL. In this case, no logging will be done.

NOTE: Stream engine does not guarantee thread safety on *custom_log*. If thread safety is a requirement, it should be satisfied in the implementation of *custom_log*.

Return value

If API instance creation was successful, tobii_api_create returns **TOBII_ERROR_NO_ERROR**. If creation failed, tobii_api_create returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *api* parameter was passed in as NULL, or the *custom_alloc* parameter was provided (it was not NULL), but one or more of its function pointers was NULL. If a custom allocator is provided, both functions (malloc_func and free_func) must be specified. Or the *custom_log* parameter was provided (it was not NULL), but the function pointer log_func was NULL. If a custom log i provided, log_func must be specified.

- **TOBII_ERROR_ALLOCATION_FAILED**

  The internal call to malloc or to the custom memory allocator (if used) returned NULL, so api creation failed.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

See also

tobii_api_destroy(), tobii_device_create()

Example

```
#include <tobii/tobii.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

// we will use custom_alloc to track allocations
typedef struct allocation_tracking
{
    int total_allocations;
    int current_allocations;
} allocation_tracking;
```

```
void* custom_malloc( void* mem_context, size_t size )
{
    allocation_tracking* tracking = (allocation_tracking*)mem_context;
    // both total allocations, and current allocations increase
    tracking->total_allocations++;
    tracking->current_allocations++;
    return malloc( size ); // pass through to C runtime
}

void custom_free( void* mem_context, void* ptr )
{
    allocation_tracking* tracking = (allocation_tracking*)mem_context;
    // only current allocations decrease, as free doesn't affect our total count
    tracking->current_allocations--;
    free( ptr ); // pass through to C runtime
}

void custom_logging( void* log_context, tobii_log_level_t level, char const* text )
{
    // log messages can be filtered by log level if desired
    if( level == TOBII_LOG_LEVEL_ERROR )
        printf( "[%d] %s\n", (int) level, text );
}

int main()
{
    allocation_tracking tracking;
    tracking.total_allocations = 0;
    tracking.current_allocations = 0;

    tobii_custom_alloc_t custom_alloc;
    custom_alloc.mem_context = &tracking;
    custom_alloc.malloc_func = &custom_malloc;
    custom_alloc.free_func = &custom_free;

    tobii_custom_log_t custom_log;
    custom_log.log_context = NULL; // we don't use the log_context in this example
    custom_log.log_func = &custom_logging;

    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, &custom_alloc, &custom_log );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );

    printf( "Total allocations: %d\n", tracking.total_allocations );
    printf( "Current allocations: %d\n", tracking.current_allocations );

    return 0;
}
```

tobii_api_destroy

Function Destroys an API instance.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_api_destroy( tobii_api_t* api );
```

Remarks When creating an instance with tobii_api_create, some system resources are acquired. When finished using the API (typically during the shutdown process), tobii_api_destroy should be called to destroy the instance and ensure that those resources are released.

tobii_api_destroy should only be called if tobii_api_create completed successfully.

api must be a pointer to a valid tobii_api_t instance as created by calling tobii_api_create.

Return value If the call was successful, tobii_api_destroy returns **TOBII_ERROR_NO_ERROR** otherwise it can return one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

The *api* parameter was passed in as NULL.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_api_destroy from within a callback function is not supported.

See also tobii_api_create(), tobii_device_destroy()

Example See tobii_api_create()

tobii_enumerate_local_device_urls

Function Retrieves the URLs for stream engine compatible devices currently connected to the system.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_enumerate_local_device_urls( tobii_api_t* api,
    tobii_device_url_receiver_t receiver, void* user_data );
```

Remarks A system might have multiple devices connected, which the stream engine is able to communicate with. tobii_enumerate_local_device_urls iterates over all such (excluding IS1 and IS2) devices found. It will only enumerate devices connected directly to the system, not devices connected on the network.

api must be a pointer to a valid tobii_api_t instance as created by calling tobii_api_create.

receiver is a function pointer to a function with the prototype:

```
void url_receiver( char const* url, void* user_data )
```

This function will be called for each device found during enumeration. It is called with the following parameters:

- *url* The URL string for the device, zero terminated. This pointer will be invalid after returning from the function, so ensure you make a copy of the string rather than storing the pointer directly.
- *user_data* This is the custom pointer sent in to tobii_enumerate_local_device_urls.

user_data custom pointer which will be passed unmodified to the receiver function.

Return value If the enumeration is successful, tobii_enumerate_local_device_urls returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_enumerate_local_device_urls returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *api* or *receiver* parameters has been passed in as NULL.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

See also tobii_device_create(), tobii_enumerate_local_device_urls_ex()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

void url_receiver( char const* url, void* user_data )
{
```

```
    int* count = (int*) user_data;
    ++(*count);
    printf( "%d. %s\n", *count, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    int count = 0;
    error = tobii_enumerate_local_device_urls( api, url_receiver, &count );
    if( error == TOBII_ERROR_NO_ERROR )
        printf( "Found %d devices.\n", count );
    else
        printf( "Enumeration failed.\n" );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_enumerate_local_device_urls_ex

Function

Retrieves the URLs for the stream engine compatible devices, of the specified generation, currently connected to the system.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_enumerate_local_device_urls_ex( tobii_api_t* api,
    tobii_device_url_receiver_t receiver, void* user_data,
    uint32_t device_generations );
```

Remarks

A system might have multiple devices connected, which the stream engine is able to communicate with. tobii_enumerate_local_device_urls_ex works similar to tobii_enumerate_local_device_urls(), but allows for more control. It only iterates over devices of the specified hardware generations, allowing for limiting the results and the processing required to enumerate devices which are not of interest for the application. It will only enumerate devices connected directly to the system, not devices connected on the network.

api must be a pointer to a valid tobii_api_t instance as created by calling tobii_api_create.

receiver is a function pointer to a function with the prototype:

```
void url_receiver( char const* url, void* user_data )
```

This function will be called for each device found during enumeration. It is called with the following parameters:

- *url* The URL string for the device, zero terminated. This pointer will be invalid after returning from the function, so ensure you make a copy of the string rather than storing the pointer directly.
- *user_data* This is the custom pointer sent in to tobii_enumerate_local_device_urls_ex.

user_data custom pointer which will be passed unmodified to the receiver function.

device_generations is a bit-field specifying which hardware generations are to be included in the enumeration. It is created by bitwise OR-ing of the following constants:

- TOBII_DEVICE_GENERATION_G5
- TOBII_DEVICE_GENERATION_IS3
- TOBII_DEVICE_GENERATION_IS4

Return value

If the enumeration is successful, tobii_enumerate_local_device_urls_ex returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_enumerate_local_device_urls_ex returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

The *api* or *receiver* parameters was passed in as NULL, or the *device_generations* parameter was passed in as 0. At least one generation must be selected for enumeration.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

See also tobii_device_create(), tobii_enumerate_local_device_urls()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

void url_receiver( char const* url, void* user_data )
{
    int* count = (int*) user_data;
    ++(*count);
    printf( "%d. %s\n", *count, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    int count = 0;
    error = tobii_enumerate_local_device_urls_ex( api, url_receiver, &count,
        TOBII_DEVICE_GENERATION_G5 | TOBII_DEVICE_GENERATION_IS4 );
    if( error == TOBII_ERROR_NO_ERROR )
        printf( "Found %d devices.\n", count );
    else
        printf( "Enumeration failed.\n" );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_device_create

Function Creates a device instance to be used for communicating with a specific device.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_device_create( tobii_api_t* api,
    char const* url, tobii_device_t** device );
```

Remarks In order to communicate with a specific device, stream engine needs to keep track of internal states. tobii_device_create allocates and initializes this state, and is needed for all functions which communicates with a device. Creating a device will establish a connection to the tracker, and can be used to query the device for more information.

api must be a pointer to a valid tobii_api_t as created by calling tobii_api_create.

url must be a valid device url as returned by tobii_enumerate_local_device_urls.

device must be a pointer to a variable of the type tobii_device_t* that is, a pointer to a tobii_device_t-pointer. This variable will be filled in with a pointer to the created device instance. tobii_device_t is an opaque type.

Return value If the device is successfully created, tobii_device_create returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_device_create returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *api* or *device* parameters were passed in as NULL, or the url string is not a valid device url (or NULL).

- **TOBII_ERROR_ALLOCATION_FAILED**

The internal call to malloc or to the custom memory allocator (if used) returned NULL, so device creation failed.

- **TOBII_ERROR_CONNECTION_FAILED**

  The connection to the device was lost. Call tobii_device_reconnect() to re-establish connection.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_device_create from within a callback function is not supported.

See also tobii_device_destroy(), tobii_enumerate_local_device_urls(), tobii_api_create(), tobii_get_device_info(), tobii_get_feature_group()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    // --> code to use the device would go here <--

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_device_destroy

Function Destroy a device previously created through a call to tobii_device_create.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_device_destroy( tobii_device_t* device );
```

Remarks tobii_device_destroy will disconnect from the device, perform cleanup and free the memory allocated by calling tobii_device_create.

NOTE: Make sure that no background thread is using the device, for example in the thread calling

tobii_device_process_callbacks, before calling tobii_device_destroy in order to avoid the risk of encountering undefined behavior.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

Return value If the device is successfully destroyed, tobii_device_create returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_device_create returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_device_destroy from within a callback function is not supported.

See also tobii_device_create(), tobii_device_create_ex()

Example See tobii_device_create()

tobii_wait_for_callbacks

Function Puts the calling thread to sleep until there are new callbacks available to process.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_wait_for_callbacks( tobii_engine_t* engine,
    int device_count, tobii_device_t* const* devices )
```

Remarks Stream engine does not use any threads to do processing or receive data. Instead, the functions tobii_device_process_callbacks() and tobii_device_process_callbacks() have to be called regularly, to receive data from the device and from Tobii Engine, and process it.

The typical use case is to implement your own thread to call tobii_device_process_callbacks and tobii_engine_process_callbacks from, and to avoid busy-waiting for data to become available, tobii_wait_for_callbacks can be called before each call to tobii_device_process_callbacks and tobii_engine_process_callbacks. It will sleep the calling thread until new data is available to process, after which tobii_device_process_callbacks and tobii_engine_process_callbacks should be called to process it.

In addition to waiting for data, tobii_wait_for_callbacks will also periodically call tobii_update_timesync() to ensure synchronization of system and device timestamps. This means you will not have to call tobii_update_timesync() if you regularly call tobii_wait_for_callbacks.

tobii_wait_for_callbacks will not wait indefinitely. There is a timeout of some hundred milliseconds, after which tobii_wait_for_callbacks will return **TOBII_ERROR_TIMED_OUT**. This does not indicate a failure - it is given as an opportunity for the calling thread to perform its own internal housekeeping (like checking for exit conditions and the like). It is valid to immediately call tobii_wait_for_callbacks again to resume waiting.

engine is a pointer to a valid tobii_engine_t instance as created by calling tobii_engine_create. It can be passed in as NULL if there is no tobii_engine_t instance.

device_count must be the number of devices in the array passed in the *devices* parameter.

devices should be an array of pointers to valid tobii_device_t instances as created by calling tobii_device_create or tobii_device_create_ex. It can be NULL if there are no tobii_device_t instances to process. In this case, *device_count* must be 0.

If the operation is successful, tobii_wait_for_callbacks returns **TOBII_ERROR_NO_ERROR**. If

Return value the call fails, or if the wait times out, tobii_wait_for_callbacks returns one of the following:

- **TOBII_ERROR_TIMED_OUT**

  This does not indicate a failure. A timeout happened before any data was received. Call tobii_wait_for_callbacks() again (it is not necessary to call tobii_device_process_callbacks() or tobii_engine_process_callbacks(), as it doesn't have any new data to process).

- **TOBII_ERROR_INVALID_PARAMETER**

  No valid device or engine instance was provided. At least one valid pointer to a device or engine instance must be provided.

- **TOBII_ERROR_CONFLICTING_API_INSTANCES**

  Every instance of device or engine passed in must be created with the same instance of tobii_api_t. If different api instances where used, this error will be returned.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

See also tobii_device_process_callbacks(), tobii_engine_process_callbacks(),

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( NULL, 1, &device );
        assert( error == TOBII_ERROR_NO_ERROR || error == TOBII_ERROR_TIMED_OUT );

        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );

    return 0;
}
```

Function

Receives data packages from the device, and sends the data through any registered callbacks.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_device_process_callbacks( tobii_device_t* device );
```

Remarks

Stream engine does not do any kind of background processing, it doesn't start any threads. It doesn't use any asynchronous callbacks. This means that in order to receive data from the device, the application needs to manually request the callbacks to happen synchronously, and this is done by calling tobii_device_process_callbacks.

tobii_device_process_callbacks will receive any data packages that are incoming from the device, process them and call any subscribed callbacks with the data. No callbacks will be called outside of tobii_device_process_callbacks, so the application have full control over when to receive callbacks.

tobii_device_process_callbacks will not wait for data, and will early-out if there's nothing to process. In order to maintain the connection to the device, tobii_device_process_callbacks should be called at least 10 times per second.

The recommended way to use tobii_device_process_callbacks, is to start a dedicated thread, and alternately call tobii_wait_for_callbacks and tobii_device_process_callbacks. See tobii_wait_for_callbacks() for more details.

If there is already a suitable thread to regularly run tobii_device_process_callbacks from (possibly interleaved with application specific operations), it is possible to do this without calling tobii_wait_for_callbacks(). In this scenario, time synchronization needs to be handled manually or the timestamps will start drifting. See tobii_update_timesync() for more details.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

Return value

If the operation is successful, tobii_device_process_callbacks returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_device_process_callbacks returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_ALLOCATION_FAILED**

  The internal call to malloc or to the custom memory allocator (if used) returned NULL, so device creation failed.

- **TOBII_ERROR_CONNECTION_FAILED**

  The connection to the device was lost. Call tobii_device_reconnect() to re-establish connection.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_device_process_callbacks from within a callback function is not supported.

See also

tobii_wait_for_callbacks(), tobii_device_clear_callback_buffers(), tobii_device_reconnect(), tobii_update_timesync()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
```

```
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        // other parts of main loop would be executed here

        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );

    return 0;
}
```

tobii_device_clear_callback_buffers

Function

Removes all unprocessed entries from the callback queues.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_device_clear_callback_buffers( tobii_device_t* device );
```

Remarks

All the data that is received and processed are written into internal buffers used for the callbacks. In some circumstances, for example during initialization, you might want to discard any data that has been buffered but not processed, without having to destroy/recreate the device, and without having to implement the filtering out of unwanted data. tobii_device_clear_callback_buffers will clear all buffered data, and only data arriving *after* the call to tobii_device_clear_callback_buffers will be forwarded to callbacks.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

Return value

If the operation is successful, tobii_device_clear_callback_buffers returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_device_clear_callback_buffers returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_device_clear_callback_buffers from within a callback function is not supported.

See also

tobii_wait_for_callbacks(), tobii_device_process_callbacks()

tobii_device_reconnect

Function Establish a new connection after a disconnect.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_device_reconnect( tobii_device_t* device );
```

Remarks When receiving the error code TOBII_ERROR_CONNECTION_FAILED, it is necessary to explicitly request reconnection, by calling tobii_device_reconnect.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

Return value

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_CONNECTION_FAILED**

  When attempting to reconnect, a connection could not be established. You might want to wait for a bit and try again, for a few times, and if the problem persists, display a message for the user.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_device_reconnect from within a callback function is not supported.

See also tobii_device_process_callbacks()

Example See tobii_device_process_callbacks()

tobii_update_timesync

Function Makes a manual re-synchronization of system timestamps and device timestamps.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_update_timesync( tobii_device_t* device );
```

Remarks The clock on the device and the clock on the system it is connected to may drift over time, and therefore they need to be periodically re-synchronized. In the default usage scenario, when regularly calling tobii_wait_for_callbacks(), this re-sychronization is handled automatically at a pre-determined interval. When not using tobii_wait_for_callbacks, and instead relying on only tobii_device_process_callbacks, it is necessary to re-synchronize manually, which is done by calling tobii_update_timesync every ~30 seconds.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

Return value If the call to tobii_update_timesync is successful, tobii_update_timesync returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_update_timesync returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_OPERATION_FAILED**

Timesync operation could not be performed at this time. Please wait a while and try again.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_update_timesync from within a callback function is not supported.

See also tobii_wait_for_callbacks(), tobii_device_reconnect(), tobii_device_process_callbacks(), tobii_system_clock()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );

        error = tobii_update_timesync( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_system_clock

Function Returns the current system time, from the same clock used to time-stamp callback data.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_system_clock( tobii_api_t* api, int64_t* timestamp_us );
```

Remarks Many of the data streams provided by the stream engine API, contains a timestamp value, measured in microseconds (us). The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values. To facilitate making comparisons between

stream engine provided timestamps and application specific events, tobii_system_clock can be used to retrieve a timestamp using the same clock and same relative values as the timestamps used in stream engine callbacks.

api must be a pointer to a valid tobii_api_t instance as created by calling tobii_api_create.

timestamp_us must be a pointer to a int64_t variable to receive the timestamp value.

Return value

If the operation is successful, tobii_system_clock returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_system_clock returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *api* or *timestamp_us* parameters were passed in as NULL.

See also

tobii_api_create()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <inttypes.h>
#include <assert.h>

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    int64_t time;
    error = tobii_system_clock( api, &time );
    if( error == TOBII_ERROR_NO_ERROR )
        printf( "timestamp: %" PRId64 "\n", time );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );

    return 0;
}
```

tobii_get_device_info

Function

Retrieves detailed information about the device, such as name and serial number.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_get_device_info( tobii_device_t* device,
    tobii_device_info_t* device_info );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

device_info is a pointer to a tobii_device_info_t variable to receive the information. It contains the following fields, all containing zero-terminated ASCII strings:

- *serial_number* the unique serial number of the device.
- *model* the model identifier for the device.
- *generation* the hardware generation, such as G5, IS3 or IS4, of the device.
- *firmware_version* the version number of the software currently installed on the device.

Return value

If device info was successfully retrieved, tobii_get_device_info returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_get_device_info returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  One or more of the *device* and *device_info* parameters were passed in as NULL.

- **TOBII_ERROR_CONNECTION_FAILED**

  The connection to the device was lost. Call tobii_device_reconnect() to re-establish connection.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_get_device_info from within a callback function is not supported.

See also tobii_device_create(), tobii_enumerate_local_device_urls()

Example

```
#include <tobii/tobii.h>
#include <assert.h>
#include <stdio.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_device_info_t info;
    error = tobii_get_device_info( device, &info );
    assert( error == TOBII_ERROR_NO_ERROR );

    printf( "Serial number: %s\n", info.serial_number );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_get_track_box

Function Retrieves 3d coordinates of the track box frustum, given in millimeters from the device center.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_get_track_box( tobii_device_t* device, tobii_track_box_t* track_box );
```

Remarks The track box is a volume in front of the tracker within which the user can be tracked.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

track_box is a pointer to a tobii_track_box_t variable to receive the result. It contains the following fields, all being arrays of three floating point values, describing the track box frustum:

- *front_upper_right_xyz, front_upper_left_xyz, front_lower_left_xyz, front_lower_right_xyz*

The four points on the frustum plane closest to the device.

- *back_upper_right_xyz*, *back_upper_left_xyz*, *back_lower_left_xyz*, *back_lower_right_xyz*

  The four points on the frustum plane furthest from the device.

Return value

If track box coordinates were successfully retrieved, tobii_get_track_box returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_get_track_box returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  One or more of the *device* and *track_box* parameters were passed in as NULL.

- **TOBII_ERROR_CONNECTION_FAILED**

  The connection to the device was lost. Call tobii_device_reconnect() to re-establish connection.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_get_track_box from within a callback function is not supported.

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_track_box_t track_box;
    error = tobii_get_track_box( device, &track_box );
    assert( error == TOBII_ERROR_NO_ERROR );

    // print just a couple of values of the track box data
    printf( "Front upper left is (%f, %f, %f)\n",
        track_box.front_upper_left_xyz[ 0 ],
        track_box.front_upper_left_xyz[ 1 ],
        track_box.front_upper_left_xyz[ 2 ] );
    printf( "Back lower right is (%f, %f, %f)\n",
        track_box.back_lower_right_xyz[ 0 ],
        track_box.back_lower_right_xyz[ 1 ],
        track_box.back_lower_right_xyz[ 2 ] );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
```

```
    return 0;
}
```

tobii_get_state_bool

Function Gets the current value of a state in the tracker.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_get_state_bool( tobii_device_t* device, tobii_state_t state,
    tobii_state_bool_t* value );
```

Remarks *device* must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

state is one of the enum values in tobii_state_t:

- **TOBII_STATE_POWER_SAVE_ACTIVE**

  Is the power save feature active on the device. This does not necessarily mean power saving measures have been engaged.

- **TOBII_STATE_REMOTE_WAKE_ACTIVE**

  Is the remote wake feature active on the device.

- **TOBII_STATE_DEVICE_PAUSED**

  Is the device paused. A paused device will keep the connection open but will not send any data while paused. This can indicate that the user temporarily wants to disable the device.

- **TOBII_STATE_EXCLUSIVE_MODE**

  Is the device in an exclusive mode. Similar to TOBII_STATE_DEVICE_PAUSED but the device is sending data to a client with exclusive access. This state is only true for short durations and does not normally need to be handled in a normal application.

value must be a pointer to a valid tobii_state_bool_t instance. On success, *value* will be set to **TOBII_STATE_BOOL_TRUE** if the state is true, otherwise **TOBII_STATE_BOOL_FALSE**. *value* will remain unmodified if the call failed.

NOTE: This method relies on cached values which is updated when tobii_device_process_callbacks() is called, so it might not represent the true state of the device if some time have passed since the last call to tobii_device_process_callbacks().

Return value If the call was successful **TOBII_ERROR_NO_ERROR** will be returned. If the call has failed one of the following error will be returned:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *value* parameter has been passed in as NULL or you passed in a *state* that is not a boolean state.

- **TOBII_ERROR_NOT_SUPPORTED**

  The device firmware has no support for retrieving the value of this state.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_get_state_bool from within a callback function is not supported.

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
```

```
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_state_bool_t value;
    error = tobii_get_state_bool( device, TOBII_STATE_DEVICE_PAUSED, &value );
    assert( error == TOBII_ERROR_NO_ERROR );

    if( value == TOBII_STATE_BOOL_TRUE )
        printf( "Device is paused!" );
    else
        printf( "Device is running!" );

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_get_state_uint32

Function

Gets the current value of a state in the tracker.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_get_state_uint32( tobii_device_t* device, tobii_state_t state,
    uint32_t* value );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

state is one of the enum values in tobii_state_t listed below:

- **TOBII_STATE_CALIBRATION_ID**

  Is the unique value identifying the calibration blob. 0 value indicates default calibration/no calibration done.

value must be a pointer to a valid uint32 instance. On success, *value* will be set to id of the calibration blob.

NOTE: This method relies on cached values which is updated when tobii_process_callbacks() is called, so it might not represent the true state of the device if some time have passed since the last call to tobii_process_callbacks().

Return value

If the call was successful **TOBII_ERROR_NO_ERROR** will be returned. If the call has failed one of the following error will be returned:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *value* parameter has been passed in as NULL or you passed in a *state* that is not a uint32 state i.e TOBII_STATE_FAULT.

- **TOBII_ERROR_NOT_SUPPORTED**

  The device firmware has no support for retrieving the value of this state.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_get_state_uint32 from within a callback function is not supported.

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <inttypes.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    uint32_t value;
    error = tobii_get_state_uint32( device, TOBII_STATE_DEVICE_PAUSED, &value );
    assert( error == TOBII_ERROR_NO_ERROR );

    printf( "%" PRIu32 "\n", value );

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_get_state_string

Function

Gets the current string value of a state in the tracker.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_get_state_string( tobii_device_t* device, tobii_state_t state,
    tobii_state_string_t value );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

state is one of the enum values in tobii_state_t listed below:

- **TOBII_STATE_FAULT**

  Retrieves a comma separated list of critical errors, if no errors exists the string “ok” is returned. If a critical error has occured the device will be unable to track or accept subscriptions.

- **TOBII_STATE_WARNING**

  Retrieves a comma separated list of warnings, if no warnings exists the string “ok” is returned. If a warning has occured the device should still be able to track and accept subscriptions.

value must be a pointer to a valid tobii_state_string_t instance. On success, *value* will be set to a null terminated string containing a maximum of 512 characters including the null termination. On failure, *value* parameter remains untouched.

NOTE: This method relies on cached values which is updated when tobii_process_callbacks() is called, so it might not represent the true state of the device if some time have passed since the last call to tobii_process_callbacks().

Return value

If the call was successful **TOBII_ERROR_NO_ERROR** will be returned. If the call has failed one of the following error will be returned:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *value* parameter has been passed in as NULL or you passed in a *state* that is not a string state i.e TOBII_STATE_CALIBRATION_ID.

- **TOBII_ERROR_NOT_SUPPORTED**

  The device firmware has no support for retrieving the value of this state.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_get_state_string from within a callback function is not supported.

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <inttypes.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_state_string_t value;
    error = tobii_get_state_string( device, TOBII_STATE_FAULT, value );
    assert( error == TOBII_ERROR_NO_ERROR );

    printf( "Device fault status: %s\n", value );

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_capability_supported

Function

Ask if a specific feature is supported or not.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_capability_supported( tobii_device_t* device,
    tobii_capability_t capability, tobii_supported_t* supported );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

capability is one of the enum values in tobii_capability_t:

- **TOBII_CAPABILITY_DISPLAY_AREA_WRITABLE**

  Query if the display area of the display can be changed by calling tobii_set_display_area().

- **TOBII_CAPABILITY_CALIBRATION_2D**

  Query if the device supports performing 2D calibration by calling tobii_calibration_collect_data_2d().

- **TOBII_CAPABILITY_CALIBRATION_3D**

  Query if the device supports performing 3D calibration by calling tobii_calibration_collect_data_3d().

- **TOBII_CAPABILITY_PERSISTENT_STORAGE**

  Query if the device supports persistent storage, needed to use tobii_license_key_store and tobii_license_key_retrieve.

- **TOBII_CAPABILITY_CALIBRATION_PER_EYE**

  Query if the device supports per-eye calibration, needed to use the per-eye calibration api.

- **TOBII_CAPABILITY_COMBINED_GAZE_VR**

  Query if the device supports combined gaze point in the wearable data stream.

- **TOBII_CAPABILITY_FACE_TYPE**

  Query if the device supports face type setting, needed to use tobii_get_face_type(), tobii_set_face_type() and tobii_enumerate_face_types().

supported must be a pointer to a valid tobii_supported_t instance. If tobii_capability_supported is successful, *supported* will be set to **TOBII_SUPPORTED** if the feature is supported, and **TOBII_NOT_SUPPORTED** if it is not.

Return value

If the call was successful **TOBII_ERROR_NO_ERROR** will be returned. If the call has failed one of the following error will be returned:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *supported* parameter has been passed in as NULL or you passed in an invalid enum value for *capability*.

- **TOBII_ERROR_CONNECTION_FAILED**

  The connection to the device was lost. Call tobii_device_reconnect() to re-establish connection.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_capability_supported from within a callback function is not supported.

See also

tobii_stream_supported()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_supported_t supported;
    error = tobii_capability_supported( device, TOBII_CAPABILITY_CALIBRATION_3D, &supported );
    assert( error == TOBII_ERROR_NO_ERROR );

    if( supported == TOBII_SUPPORTED )
        printf( "Device supports 3D calibration." );
    else
        printf( "Device does not support 3D calibration." );

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_stream_supported

Function

Ask if a specific stream is supported or not.

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_stream_supported( tobii_device_t* device,
    tobii_stream_t stream, tobii_supported_t* supported );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

stream is one of the enum values in tobii_stream_t, each corresponding to one of the streams from tobii_streams.h, tobii_wearable.h and tobii_advanced.h

- **TOBII_STREAM_GAZE_POINT**
- **TOBII_STREAM_GAZE_ORIGIN**
- **TOBII_STREAM_EYE_POSITION_NORMALIZED**
- **TOBII_STREAM_USER_PRESENCE**
- **TOBII_STREAM_HEAD_POSE**
- **TOBII_STREAM_WEARABLE**
- **TOBII_STREAM_GAZE_DATA**
- **TOBII_STREAM_DIGITAL_SYNCPORT**
- **TOBII_STREAM_DIAGNOSTICS_IMAGE**

supported must be a pointer to a valid tobii_supported_t instance. If tobii_stream_supported is successful, *supported* will be set to **TOBII_SUPPORTED** if the feature is supported, and **TOBII_NOT_SUPPORTED** if it is not.

Return value

If the call was successful **TOBII_ERROR_NO_ERROR** will be returned. If the call has failed one of

the following error will be returned:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *supported* parameter has been passed in as NULL or you passed in an invalid enum value for *stream*.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_stream_supported from within a callback function is not supported.

See also tobii_capability_supported()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_supported_t supported;
    error = tobii_stream_supported( device, TOBII_STREAM_GAZE_POINT, &supported );
    assert( error == TOBII_ERROR_NO_ERROR );

    if( supported == TOBII_SUPPORTED )
        printf( "Device supports gaze point stream." );
    else
        printf( "Device does not support gaze point stream." );

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_streams.h

The tobii_streams.h header file is used for managing data stream subscriptions. There are several types of data streams in the API, and tobii_streams.h contains functions to subscribe to and unsubscribe from these streams, as well as data structures describing the data packages.

Please note that there can only be one callback registered to a stream at a time. To register a new callback, first unsubscribe from the stream, then resubscribe with the new callback function.

Do NOT call StreamEngine API functions from within the callback functions, due to risk of internal deadlocks. Generally one should finish the callback functions as quickly as possible and not make any blocking calls.

tobii_gaze_point_subscribe

Function Start listening for gaze point data; the position on the screen that the user is currently looking at.

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_gaze_point_subscribe( tobii_device_t* device,
    tobii_gaze_point_callback_t callback, void* user_data );
```

Remarks This subscription is for receiving the point on the screen, in normalized (0 to 1) coordinates, that the user is currently looking at. The data is lightly filtered for stability.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

callback is a function pointer to a function with the prototype:

```
void gaze_point_callback( tobii_gaze_point_t const* gaze_point, void* user_data )
```

This function will be called when there is new gaze data available. It is called with the following parameters:

- *gaze_point*

  This is a pointer to a struct containing the following data:

  - *timestamp_us* Timestamp value for when the gaze point was captured, measured in microseconds (us). The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values. The function tobii_system_clock() can be used to retrieve a timestamp using the same clock and same relative values as this timestamp.
  - *validity* **TOBII_VALIDITY_VALID** if the gaze point is valid, **TOBII_VALIDITY_INVALID** if it is not. The value of the *position_xy* field is unspecified unless *validity* is **TOBII_VALIDITY_VALID**.
  - *position_xy* An array of two floats, for the horizontal (x) and vertical (y) screen coordinate of the gaze point. The left edge of the screen is 0.0, and the right edge is 1.0. The top edge of the screen is 0.0, and the bottom edge is 1.0. Note that the value might be outside the 0.0 to 1.0 range, if the user looks outside the screen.
- *user_data* This is the custom pointer sent in when registering the callback.

user_data custom pointer which will be passed unmodified to the callback.

Return value If the operation is successful, tobii_gaze_point_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_gaze_point_subscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *callback* parameters were passed in as NULL.
- **TOBII_ERROR_ALREADY_SUBSCRIBED**

A subscription for gaze points were already made. There can only be one callback registered at a time. To change to another callback, first call tobii_gaze_point_unsubscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with an old device and/or that is running outdated firmware.

- **TOBII_ERROR_TOO_MANY_SUBSCRIBERS**

  Too many subscribers for the requested stream. Tobii eye trackers can have a limitation on the number of concurrent subscribers to specific streams due to high bandwidth and/or high frequency of the data stream.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_gaze_point_subscribe from within a callback function is not supported.

See also tobii_gaze_point_unsubscribe(), tobii_device_process_callbacks(), tobii_system_clock()

Example

```
#include <tobii/tobii_streams.h>
#include <stdio.h>
#include <assert.h>

void gaze_point_callback( tobii_gaze_point_t const* gaze_point, void* user_data )
{
    if( gaze_point->validity == TOBII_VALIDITY_VALID )
        printf( "Gaze point: %f, %f\n",
            gaze_point->position_xy[ 0 ],
            gaze_point->position_xy[ 1 ] );
}

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_gaze_point_subscribe( device, gaze_point_callback, 0 );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( NULL, 1, &device );
        assert( error == TOBII_ERROR_NO_ERROR || error == TOBII_ERROR_TIMED_OUT );

        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }
```

```
    error = tobii_gaze_point_unsubscribe( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_gaze_point_unsubscribe

Function Stops listening to gaze point stream that was subscribed to by a call to tobii_gaze_point_subscribe()

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_gaze_point_unsubscribe( tobii_device_t* device );
```

Remarks *device* must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

Return value If the operation is successful, tobii_gaze_point_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_gaze_point_unsubscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUBSCRIBED**

  There was no subscription for gaze points. It is only valid to call tobii_gaze_point_unsubscribe() after first successfully calling tobii_gaze_point_subscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with an old device and/or that is running outdated firmware.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_gaze_point_unsubscribe from within a callback function is not supported.

See also tobii_gaze_point_subscribe()

Example See tobii_gaze_point_subscribe()

tobii_gaze_origin_subscribe

Function Start listening for gaze origin data. Gaze origin is a point on the users eye, reported in millimeters from the center of the display.

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_gaze_origin_subscribe( tobii_device_t* device,
    tobii_gaze_origin_callback_t callback, void* user_data );
```

Remarks This subscription is for receiving the origin of the gaze vector, measured in millimeters from the center of the display. Gaze origin is a point on the users eye, but the exact point of the origin

varies by device. For example, it might be defined as the center of the pupil or the center of the cornea. The data is lightly filtered for stability.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

callback is a function pointer to a function with the prototype:

```
void gaze_origin_callback( tobii_gaze_origin_t const* gaze_origin, void* user_data )
```

This function will be called when there is new gaze origin data available. It is called with the following parameters:

- *gaze_origin*

  This is a pointer to a struct containing the following data:

  - *timestamp_us* Timestamp value for when the gaze origin was calculated, measured in microseconds (us). The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values. The function tobii_system_clock() can be used to retrieve a timestamp using the same clock and same relative values as this timestamp.
  - *left_validity* **TOBII_VALIDITY_INVALID** if the values for the left eye are not valid, **TOBII_VALIDITY_VALID** if they are.
  - *left_xyz* An array of three floats, for the x, y and z coordinate of the gaze origin point on the left eye of the user, as measured in millimeters from the center of the display.
  - *right_validity* **TOBII_VALIDITY_INVALID** if the values for the right eye are not valid, **TOBII_VALIDITY_VALID** if they are.
  - *right_xyz* An array of three floats, for the x, y and z coordinate of the gaze origin point on the right eye of the user, as measured in millimeters from the center of the display.
- *user_data* This is the custom pointer sent in when registering the callback.

user_data custom pointer which will be passed unmodified to the callback.

Return value

If the operation is successful, tobii_gaze_origin_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_gaze_origin_subscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_ALREADY_SUBSCRIBED**

  A subscription for gaze origins were already made. There can only be one callback registered at a time. To change to another callback, first call tobii_gaze_origin_unsubscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with an old device and/or that is running outdated firmware.

- **TOBII_ERROR_TOO_MANY_SUBSCRIBERS**

  Too many subscribers for the requested stream. Tobii eye trackers can have a limitation on the number of concurrent subscribers to specific streams due to high bandwidth and/or high frequency of the data stream.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(),

tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_gaze_origin_subscribe from within a callback function is not supported.

See also tobii_eye_position_normalized_subscribe(), tobii_gaze_origin_unsubscribe(), tobii_device_process_callbacks(), tobii_system_clock()

Example

```
#include <tobii/tobii_streams.h>
#include <stdio.h>
#include <assert.h>

void gaze_origin_callback( tobii_gaze_origin_t const* gaze_origin, void* user_data )
{
    if( gaze_origin->left_validity == TOBII_VALIDITY_VALID )
        printf( "Left: %f, %f, %f ",
            gaze_origin->left_xyz[ 0 ],
            gaze_origin->left_xyz[ 1 ],
            gaze_origin->left_xyz[ 2 ] );

    if( gaze_origin->right_validity == TOBII_VALIDITY_VALID )
        printf( "Right: %f, %f, %f ",
            gaze_origin->right_xyz[ 0 ],
            gaze_origin->right_xyz[ 1 ],
            gaze_origin->right_xyz[ 2 ] );

    printf( "\n" );
}

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_gaze_origin_subscribe( device, gaze_origin_callback, 0 );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( NULL, 1, &device );
        assert( error == TOBII_ERROR_NO_ERROR || error == TOBII_ERROR_TIMED_OUT );

        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_gaze_origin_unsubscribe( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_gaze_origin_unsubscribe

Function Stops listening to gaze origin stream that was subscribed to by a call to tobii_gaze_origin_subscribe()

Syntax
```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_gaze_origin_unsubscribe( tobii_device_t* device );
```

Remarks *device* must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

Return value If the operation is successful, tobii_gaze_origin_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_gaze_origin_unsubscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUBSCRIBED**

  There was no subscription for gaze origins. It is only valid to call tobii_gaze_origin_unsubscribe() after first successfully calling tobii_gaze_origin_subscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with an old device and/or that is running outdated firmware.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_gaze_origin_unsubscribe from within a callback function is not supported.

See also tobii_gaze_origin_subscribe()

Example See tobii_gaze_origin_subscribe()

tobii_eye_position_normalized_subscribe

Function Start listening for normalized eye position data. Eye position is a point on the users eye, reported in normalized track box coordinates.

Syntax
```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_eye_position_normalized_subscribe( tobii_device_t* device,
    tobii_eye_position_normalized_callback_t callback, void* user_data );
```

Remarks This subscription is for receiving the position of the eyes, given in normalized (0 to 1) track box coordinates. The exact point on the eye varies by device. For example, the center of the pupil or the center of the cornea. The data is lightly filtered for stability. The track box is a the volume around the user that the device can track within.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

callback is a function pointer to a function with the prototype:

```
void eye_position_normalized_callback( tobii_eye_position_normalized_t const* eye_position,
void* user_data )
```

This function will be called when there is new normalized eye position data available. It is called with the following parameters:

- *eye_position*

  This is a pointer to a struct containing the following data:

  - *timestamp_us*

    Timestamp value for when the gaze origin was calculated, measured in microseconds (us). The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values. The function tobii_system_clock() can be used to retrieve a timestamp using the same clock and same relative values as this timestamp.

  - *left_validity*

    TOBII_VALIDITY_INVALID if the values for the left eye are not valid, **TOBII_VALIDITY_VALID** if they are.

  - *left_xyz*

    An array of three floats, for the x, y and z coordinate of the eye position on the left eye of the user, as a normalized value within the track box.

  - *right_validity*

    TOBII_VALIDITY_INVALID if the values for the right eye are not valid, **TOBII_VALIDITY_VALID** if they are.

  - *right_xyz*

    An array of three floats, for the x, y and z coordinate of the eye position on the right eye of the user, as a normalized value within the track box.

- *user_data* This is the custom pointer sent in when registering the callback.

user_data custom pointer which will be passed unmodified to the callback.

Return value

If the operation is successful, tobii_eye_position_normalized_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_eye_position_normalized_subscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *callback* parameter were passed in as NULL.

- **TOBII_ERROR_ALREADY_SUBSCRIBED**

  A subscription for normalized eye positions were already made. There can only be one callback registered at a time. To change to another callback, first call tobii_eye_position_normalized_unsubscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with an old device and/or that is running outdated firmware.

- **TOBII_ERROR_TOO_MANY_SUBSCRIBERS**

  Too many subscribers for the requested stream. Tobii eye trackers can have a limitation on the number of concurrent subscribers to specific streams due to high bandwidth and/or high frequency of the data stream.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call

such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_eye_position_normalized_subscribe from within a callback function is not supported.

See also tobii_gaze_origin_subscribe(), tobii_eye_position_normalized_unsubscribe(), tobii_device_process_callbacks(), tobii_system_clock()

Example

```
#include <tobii/tobii_streams.h>
#include <stdio.h>
#include <assert.h>

void eye_position_callback( tobii_eye_position_normalized_t const* eye_pos, void* user_data )
{
    if( eye_pos->left_validity == TOBII_VALIDITY_VALID )
        printf( "Left: %f, %f, %f ",
            eye_pos->left_xyz[ 0 ],
            eye_pos->left_xyz[ 1 ],
            eye_pos->left_xyz[ 2 ] );

    if( eye_pos->right_validity == TOBII_VALIDITY_VALID )
        printf( "Right: %f, %f, %f ",
            eye_pos->right_xyz[ 0 ],
            eye_pos->right_xyz[ 1 ],
            eye_pos->right_xyz[ 2 ] );

    printf( "\n" );
}

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_eye_position_normalized_subscribe( device, eye_position_callback, 0 );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( NULL, 1, &device );
        assert( error == TOBII_ERROR_NO_ERROR || error == TOBII_ERROR_TIMED_OUT );

        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_eye_position_normalized_unsubscribe( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_eye_position_normalized_unsubscribe

Function Stops listening to normalized eye position stream that was subscribed to by a call to tobii_eye_position_normalized_subscribe()

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_eye_position_normalized_unsubscribe(
    tobii_device_t* device );
```

Remarks *device* must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

Return value If the operation is successful, tobii_eye_position_normalized_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_eye_position_normalized_unsubscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUBSCRIBED**

  There was no subscription for normalized eye positions. It is only valid to call tobii_eye_position_normalized_unsubscribe() after first successfully calling tobii_eye_position_normalized_subscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with an old device and/or that is running outdated firmware.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_eye_position_normalized_unsubscribe from within a callback function is not supported.

See also tobii_eye_position_normalized_subscribe()

Example See tobii_eye_position_normalized_subscribe()

tobii_user_presence_subscribe

Function Start listening for user presence notifications, reporting whether there is a person in front of the device.

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_user_presence_subscribe( tobii_device_t* device,
    tobii_user_presence_callback_t callback, void* user_data );
```

Remarks This subscription is for being notified when a user is detected by the device, and when a user is no longer detected.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

callback is a function pointer to a function with the prototype:

```
void presence_callback( tobii_user_presence_status_t status, int64_t timestamp_us,
    void* user_data )
```

This function will be called when there is a change in presence state. It is called with the following parameters:

- *status* One of the following values:
  - **TOBII_USER_PRESENCE_STATUS_UNKNOWN** if user presence could not be determined.
  - **TOBII_USER_PRESENCE_STATUS_AWAY** if there is a user in front of the device.
  - **TOBII_USER_PRESENCE_STATUS_PRESENT** if there is no user in front of the device.
- *timestamp_us* Timestamp value for when the user presence was calculated, measured in microseconds (us). The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values. The function tobii_system_clock() can be used to retrieve a timestamp using the same clock and same relative values as this timestamp.
- *user_data* This is the custom pointer sent in when registering the callback.

user_data custom pointer which will be passed unmodified to the callback.

Return value

If the operation is successful, tobii_user_presence_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_user_presence_subscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *callback* parameter were passed in as NULL.

- **TOBII_ERROR_ALREADY_SUBSCRIBED**

  A subscription for presence data was already made. There can only be one callback registered at a time. To change to another callback, first call tobii_user_presence_unsubscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with an old device and/or that is running outdated firmware.

- **TOBII_ERROR_TOO_MANY_SUBSCRIBERS**

  Too many subscribers for the requested stream. Tobii eye trackers can have a limitation on the number of concurrent subscribers to specific streams due to high bandwidth and/or high frequency of the data stream.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_user_presence_subscribe from within a callback function is not supported.

See also

tobii_user_presence_unsubscribe(), tobii_device_process_callbacks(), tobii_system_clock()

Example

```
#include <tobii/tobii_streams.h>
#include <stdio.h>
#include <assert.h>

void presence_callback( tobii_user_presence_status_t status, int64_t timestamp_us, void*
user_data )
{
    switch( status )
    {
        case TOBII_USER_PRESENCE_STATUS_UNKNOWN:
            printf( "User presence status is unknown.\n" );
            break;
        case TOBII_USER_PRESENCE_STATUS_AWAY:
```

```
            printf( "User is away.\n" );
            break;
        case TOBII_USER_PRESENCE_STATUS_PRESENT:
            printf( "User is present.\n" );
            break;
    }
}

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_user_presence_subscribe( device, presence_callback, 0 );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( NULL, 1, &device );
        assert( error == TOBII_ERROR_NO_ERROR || error == TOBII_ERROR_TIMED_OUT );

        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_user_presence_unsubscribe( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_user_presence_unsubscribe

Function

Stops listening to presence stream that was subscribed to by a call to tobii_user_presence_subscribe().

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_user_presence_unsubscribe( tobii_device_t* device );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

Return value

If the operation is successful, tobii_user_presence_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_user_presence_unsubscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUBSCRIBED**

  There was no subscription for presence. It is only valid to call tobii_user_presence_unsubscribe() after first successfully calling tobii_user_presence_subscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with an old device and/or that is running outdated firmware.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_user_presence_unsubscribe from within a callback function is not supported.

See also tobii_user_presence_subscribe()

Example See tobii_user_presence_subscribe()

tobii_head_pose_subscribe

Function Start listening to the head pose stream, which reports the position and rotation of the user's head.

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t TOBII_CALL tobii_head_pose_subscribe( tobii_device_t* device,
    tobii_head_pose_callback_t callback, void* user_data );
```

Remarks *device* must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

callback is a function pointer to a function with the prototype:

```
void head_pose_callback( tobii_head_pose_t const* head_pose, void* user_data )
```

This function will be called when there is new head pose data to be sent to the subscriber. It is called with the following parameters:

- *head_pose*

  This is a pointer to a struct containing the following data:

  - *timestamp_us*

    Timestamp value for when the head pose was calculated, measured in microseconds (us). The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values. The function tobii_system_clock() can be used to retrieve a timestamp using the same clock and same relative values as this timestamp.

  - *position_validity*

    Indicates the validity of the position_xyz field. **TOBII_VALIDITY_INVALID** if the field is not valid, **TOBII_VALIDITY_VALID** if it is.

  - *position_xyz*

    An array of three floats, for the x, y and z coordinate of the head of the user, as measured in millimeters from the center of the display.

  - *rotation_validity_xyz*

    An array indicating the validity of each elemnt of the rotation_xyz field.

TOBII_VALIDITY_INVALID if the element is not valid, **TOBII_VALIDITY_VALID** if it is.

- *rotation_xyz*

  An array of three floats, for the x, y and z rotation of the head of the user. The rotation is expressed in Euler angles using right-handed rotations around each axis. The z rotation describes the rotation around the vector pointing towards the user.

- *user_data*

  This is the custom pointer sent in when registering the callback.

user_data custom pointer which will be passed unmodified to the notification callback.

Return value

If the operation is successful, tobii_head_pose_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_head_pose_subscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *callback* parameter were passed in as NULL.

- **TOBII_ERROR_ALREADY_SUBSCRIBED**

  A subscription for head pose were already made. There can only be one callback registered at a time. To change to another callback, first call tobii_head_pose_unsubscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support head pose. This error is returned if the API is called with an old device which doesn't support head pose.

- **TOBII_ERROR_TOO_MANY_SUBSCRIBERS**

  Too many subscribers for the requested stream. Tobii eye trackers can have a limitation on the number of concurrent subscribers to specific streams due to high bandwidth and/or high frequency of the data stream.

- **TOBII_ERROR_NOT_AVAILABLE**

  Head pose is not available as the software component responsible for providing it is not running. Head pose requires the Tobii Eye Tracking Core Software to be installed and running.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_head_pose_subscribe from within a callback function is not supported.

See also

tobii_head_pose_unsubscribe()

Example

```
#include <tobii/tobii_streams.h>
#include <stdio.h>
#include <assert.h>

void head_pose_callback( tobii_head_pose_t const* head_pose, void* user_data )
{
    if( head_pose->position_validity == TOBII_VALIDITY_VALID )
        printf( "Position: (%f, %f, %f)\n",
            head_pose->position_xyz[ 0 ],
            head_pose->position_xyz[ 1 ],
            head_pose->position_xyz[ 2 ] );

    printf( "Rotation:\n" );
    for( int i = 0; i < 3; ++i )
        if( head_pose->rotation_validity_xyz[ i ] == TOBII_VALIDITY_VALID )
            printf( "%f\n", head_pose->rotation_xyz[ i ] );
```

```
}

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_head_pose_subscribe( device, head_pose_callback, 0 );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( NULL, 1, &device );
        assert( error == TOBII_ERROR_NO_ERROR || error == TOBII_ERROR_TIMED_OUT );

        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_head_pose_unsubscribe( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_head_pose_unsubscribe

Function

Stops listening to the head pose stream that was subscribed to by a call to tobii_head_pose_subscribe().

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t TOBII_CALL tobii_head_pose_unsubscribe( tobii_device_t* device );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

Return value

If the operation is successful, tobii_head_pose_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_head_pose_unsubscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUBSCRIBED**

  There was no subscription for head pose. It is only valid to call tobii_head_pose_unsubscribe() after first successfully calling tobii_head_pose_subscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

The device doesn't support head pose. This error is returned if the API is called with an old device which doesn't support head pose.

- **TOBII_ERROR_NOT_AVAILABLE**

  Head pose is not available as the software component responsible for providing it is not running.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_head_pose_unsubscribe from within a callback function is not supported.

See also tobii_head_pose_subscribe()

Example See tobii_head_pose_subscribe()

tobii_notifications_subscribe

Function Start listening to the notifications stream, which reports state changes for a device.

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_notifications_subscribe( tobii_device_t* device,
    tobii_notifications_callback_t callback, void* user_data );
```

Remarks As the device is a shared resource, which may be in use by multiple client applications, notifications are used to inform when a state change have occured on the device, as an effect of another client performing some operation (such as starting a calibration, or changing the display area).

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

callback is a function pointer to a function with the prototype:

```
void notification_callback( tobii_notification_t const* notification, void* user_data )
```

This function will be called when there is a new notification to be sent to the subscriber. It is called with the following parameters:

- *notification*

  This is a pointer to a struct containing the following data:

  - *type*

    Denotes the type of notification that was received. Can be one of the following values:

    TOBII_NOTIFICATION_TYPE_CALIBRATION_STATE_CHANGED
    TOBII_NOTIFICATION_TYPE_EXCLUSIVE_MODE_STATE_CHANGED
    TOBII_NOTIFICATION_TYPE_TRACK_BOX_CHANGED
    TOBII_NOTIFICATION_TYPE_DISPLAY_AREA_CHANGED
    TOBII_NOTIFICATION_TYPE_FRAMERATE_CHANGED
    TOBII_NOTIFICATION_TYPE_POWER_SAVE_STATE_CHANGED
    TOBII_NOTIFICATION_TYPE_DEVICE_PAUSED_STATE_CHANGED
    TOBII_NOTIFICATION_TYPE_CALIBRATION_ENABLED_EYE_CHANGED
    TOBII_NOTIFICATION_TYPE_CALIBRATION_ID_CHANGED
    TOBII_NOTIFICATION_TYPE_COMBINED_GAZE_FACTOR_CHANGED
    TOBII_NOTIFICATION_TYPE_FAULTS_CHANGED
    TOBII_NOTIFICATION_TYPE_WARNINGS_CHANGED
    TOBII_NOTIFICATION_TYPE_FACE_TYPE_CHANGED

- *value_type*

  Indicates which of the fields of the *value* union contains the data. Can be one of the following:

  TOBII_NOTIFICATION_VALUE_TYPE_NONE
  TOBII_NOTIFICATION_VALUE_TYPE_FLOAT
  TOBII_NOTIFICATION_VALUE_TYPE_STATE
  TOBII_NOTIFICATION_VALUE_TYPE_DISPLAY_AREA
  TOBII_NOTIFICATION_VALUE_TYPE_UINT
  TOBII_NOTIFICATION_VALUE_TYPE_ENABLED_EYE
  TOBII_NOTIFICATION_VALUE_TYPE_STRING

- *value*

  The attached data described in *value_type*, which is used to access the corresponding data field. This value is guaranteed to be related to the notification its attached to.

- *user_data*

  This is the custom pointer sent in when registering the callback.

user_data custom pointer which will be passed unmodified to the notification callback.

Return value

If the operation is successful, tobii_notifications_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_notifications_subscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *callback* parameters were passed in as NULL.

- **TOBII_ERROR_ALREADY_SUBSCRIBED**

  A subscription for notifications were already made. There can only be one callback registered at a time. To change to another callback, first call tobii_notifications_unsubscribe().

- **TOBII_ERROR_TOO_MANY_SUBSCRIBERS**

  Too many subscribers for the requested stream. Tobii eye trackers can have a limitation on the number of concurrent subscribers to specific streams due to high bandwidth and/or high frequency of the data stream.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_notifications_subscribe from within a callback function is not supported.

See also

tobii_notifications_unsubscribe(), tobii_device_process_callbacks()

Example

```
#include <tobii/tobii_streams.h>
#include <stdio.h>
#include <assert.h>

void notifications_callback( tobii_notification_t const* notification, void* user_data )
{
    if( notification->type == TOBII_NOTIFICATION_TYPE_CALIBRATION_STATE_CHANGED )
    {
        if( notification->value.state == TOBII_STATE_BOOL_TRUE )
            printf( "Calibration started\n" );
        else
            printf( "Calibration stopped\n" );
    }

    if( notification->type == TOBII_NOTIFICATION_TYPE_FRAMERATE_CHANGED )
        printf( "Framerate changed\nNew framerate: %f\n", notification->value.float_ );
```

```
}

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_notifications_subscribe( device, notifications_callback, 0 );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( NULL, 1, &device );
        assert( error == TOBII_ERROR_NO_ERROR || error == TOBII_ERROR_TIMED_OUT );

        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_notifications_unsubscribe( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_notifications_unsubscribe

Function

Stops listening to notifications stream that was subscribed to by a call to tobii_notifications_subscribe()

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_notifications_unsubscribe( tobii_device_t* device );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create.

Return value

If the operation is successful, tobii_notifications_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_notifications_unsubscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUBSCRIBED**

  There was no subscription for notifications. It is only valid to call tobii_notifications_unsubscribe() after first successfully calling tobii_notifications_subscribe().

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_notifications_unsubscribe from within a callback function is not supported.

See also tobii_notifications_subscribe()

Example See tobii_notifications_subscribe()

tobii_wearable.h

tobii_wearable.h contains functions relating to wearable devices, such as VR headsets. It contains a specialized data stream with different data from the regular streams, as well as functions to retrieve and modify the lens configuration of the device.

tobii_wearable_data_subscribe

Function Start listening for eye tracking data from wearable device, such as VR headsets.

Syntax

```
#include <tobii/tobii_wearable.h>
tobii_error_t tobii_wearable_data_subscribe( tobii_device_t* device,
    tobii_wearable_data_callback_t callback, void* user_data );
```

Remarks All coordinates are expressed in a right-handed Cartesian system.

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

callback is a function pointer to a function with the prototype:

```
void wearable_callback( tobii_wearable_data_t const* data, void* user_data )
```

This function will be called when there is new data available. It is called with the following parameters:

- *data* This is a pointer to a struct containing the data listed below. Note that it is only valid during the callback. Its data should be copied if access is necessary at a later stage, from outside the callback.
    - *timestamp_tracker_us* Timestamp value for when the data was captured, measured in microseconds (us). It is generated on the device responsible for capturing the data. The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values. The value returned in *timestamp_system_us* is calculated from this value.
    - *timestamp_system_us* Timestamp value for when the data was captured, measured in microseconds (us), and synchronized with the clock of the computer. The function tobii_system_clock can be used to retrieve a timestamp (at the time of the call) using the same clock and same relative values as this timestamp. The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values.
    - *frame_counter* A counter that increments by one each frame. There is no guarantee on its initial value. Will eventually wrap around and restart at 0, which may be necessary to detect and handle if comparing the values between frames.
    - *led_mode* A bitmask where each bit (starting from the least significant bit) represents a LED group and whether it is active or not, with a value of 1 being active and 0 inactive.
    - *left* This is a struct containing the following data, related to the left eye:
        - *gaze_origin_validity* **TOBII_VALIDITY_INVALID** if *gaze_origin_mm_xyz* is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.
        - *gaze_origin_mm_xyz* An array of three floats, for the x, y and z coordinate of the point in the user's eye from which the calculated gaze ray originates, expressed in a right-handed Cartesian coordinate system. See the wearable hardware specification for its origin.
        - *gaze_direction_validity* **TOBII_VALIDITY_INVALID** if *gaze_direction_normalized_xyz* for the eye is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.

- *gaze_direction_normalized_xyz* An array of three floats, for the x, y and z coordinate of the gaze direction of the eye of the user, expressed as a unit vector in a right-handed Cartesian coordinate system.
- *pupil_diameter_validity* **TOBII_VALIDITY_INVALID** if *pupil_diameter_mm* is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.
- *pupil_diameter_mm* A float that represents the approximate diameter of the pupil, expressed in millimeters. Only relative changes are guaranteed to be accurate.
- *eye_openness_validity* **TOBII_VALIDITY_INVALID** if *eye_openess* for the eye is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.
- *eye_openness* A float that represents how open the user's eye is, defined as the ratio between the height of the eye divided by its width, making a fully open eye yield a value of approximately 0.5.
- *pupil_position_in_sensor_area_validity* **TOBII_VALIDITY_INVALID** if *pupil_position_in_sensor_area_xy* is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.
- *pupil_position_in_sensor_area_xy* An array of two floats, for the x and y of the position of the pupil normalized to the sensor area where (0, 0): is the top left of sensor area, from the sensor's perspective (1, 1): is the bottom right of sensor area, from the sensor's perspective In systems where multiple cameras observe both eyes, this signal gives the pupil position in the primary sensor. Useful for detecting and visualizing how well the eyes are centered in the sensor images.

- *right* This is another instance of the same struct as in *left*, but which holds data related to the right eye of the user.
- *gaze_origin_combined_validity* **TOBII_VALIDITY_INVALID** if *gaze_origin_combined_mm_xyz* is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.

  This field will only be set if you have the capability TOBII_CAPABILITY_COMBINED_GAZE_VR. See tobii_capability_supported().
- *gaze_origin_combined_mm_xyz* An array of three floats, for the x, y and z coordinate of the point in from which the combined gaze ray originates, expressed in a right-handed Cartesian coordinate system.

  This field will only be set if you have the capability TOBII_CAPABILITY_COMBINED_GAZE_VR. See tobii_capability_supported().
- *gaze_direction_combined_validity* **TOBII_VALIDITY_INVALID** if *gaze_direction_combined_normalized_xyz* is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.

  This field will only be set if you have the capability TOBII_CAPABILITY_COMBINED_GAZE_VR. See tobii_capability_supported().
- *gaze_direction_combined_normalized_xyz* An array of three floats, for the x, y and z coordinate of the combined gaze direction of the left and right eye of the user, expressed as a unit vector in a right-handed Cartesian coordinate system.

  This field will only be set if you have the capability TOBII_CAPABILITY_COMBINED_GAZE_VR. See tobii_capability_supported().

- *user_data* This is the custom pointer sent in when registering the callback.

user_data custom pointer which will be passed unmodified to the callback function.

Return value

If the operation is successful, tobii_wearable_data_subscribe() returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_wearable_data_subscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  One or more of the *device* and *callback* parameters were passed in as NULL.

- **TOBII_ERROR_ALREADY_SUBSCRIBED**

  A subscription for wearable data were already made. There can only be one callback registered at a time. To change to another callback, first call tobii_wearable_data_unsubscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with a non-VR device.

- **TOBII_ERROR_TOO_MANY_SUBSCRIBERS**

  Too many subscribers for the requested stream. Tobii eye trackers can have a limitation on the number of concurrent subscribers to specific streams due to high bandwidth and/or high frequency of the data stream.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_wearable_data_subscribe from within a callback function is not supported.

See also tobii_wearable_data_unsubscribe(), tobii_device_process_callbacks(), tobii_capability_supported()

Example

```
#include <tobii/tobii_wearable.h>
#include <stdio.h>
#include <assert.h>

void wearable_callback( tobii_wearable_data_t const* wearable,
    void* user_data )
{
    if( wearable->left.gaze_direction_validity )
    {
        printf( "Left gaze direction: (%f, %f, %f)\n",
            wearable->left.gaze_direction_normalized_xyz[ 0 ],
            wearable->left.gaze_direction_normalized_xyz[ 1 ],
            wearable->left.gaze_direction_normalized_xyz[ 2 ] );
    }
    else
        printf( "Left gaze direction: INVALID\n" );

    if( wearable->right.gaze_direction_validity )
    {
        printf( "Right gaze direction: (%f, %f, %f)\n",
            wearable->right.gaze_direction_normalized_xyz[ 0 ],
            wearable->right.gaze_direction_normalized_xyz[ 1 ],
            wearable->right.gaze_direction_normalized_xyz[ 2 ] );
    }
    else
        printf( "Right gaze direction: INVALID\n" );
}

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
```

```
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_wearable_data_subscribe( device, wearable_callback, 0 );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( NULL, 1, &device );
        assert( error == TOBII_ERROR_NO_ERROR || error == TOBII_ERROR_TIMED_OUT );

        error = tobii_device_process_callbacks( device );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_wearable_data_unsubscribe( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_wearable_data_unsubscribe

Function

Stops listening to the wearable data stream that was subscribed to by a call to tobii_wearable_data_subscribe().

Syntax

```
#include <tobii/tobii_wearable.h>
tobii_error_t TOBII_CALL tobii_wearable_data_unsubscribe( tobii_device_t* device );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

Return value

If the operation is successful, tobii_wearable_data_unsubscribe() returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_wearable_data_unsubscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUBSCRIBED**

  There was no subscription for wearable data. It is only valid to call tobii_wearable_data_unsubscribe() after first successfully calling tobii_wearable_data_subscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support the stream. This error is returned if the API is called with an old device and/or that is running outdated firmware.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling

tobii_wearable_data_unsubscribe from within a callback function is not supported.

See also tobii_wearable_data_subscribe()

tobii_get_lens_configuration

Function Retrieves the current lens configuration in the tracker.

Syntax

```
#include <tobii/tobii_wearable.h>
tobii_error_t TOBII_CALL tobii_get_lens_configuration( tobii_device_t* device,
    tobii_lens_configuration_t* lens_config );
```

Remarks *device* must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

lens_config must be a pointer to a valid tobii_lens_configuration_t. Upon success, it will be populated with the relevant data. It will remain unmodified upon failure. It is a pointer to a struct containing the following data:

- *left* An array of three floats, for the x, y and z offset of the left lens in the headset, given in millimeters.
- *right* An array of three floats, for the x, y and z offset of the right lens in the headset, given in millimeters.

Return value If the operation is successful, tobii_get_lens_configuration() returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_get_lens_configuration returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *lens_config* parameter was passed in as NULL.

- **TOBII_ERROR_CONNECTION_FAILED**

  The connection to the device was lost. Call tobii_device_reconnect() to re-establish connection.

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support this functionality. This error is returned if the API is called with a non-VR device.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_get_lens_configuration from within a callback function is not supported.

See also tobii_set_lens_configuration()

Example

```
#include <tobii/tobii_wearable.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
```

```
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_lens_configuration_t lens_config;
    error = tobii_get_lens_configuration( device, &lens_config );
    assert( error == TOBII_ERROR_NO_ERROR );

    printf( "VR lens offset (left): (%f, %f, %f)\n",
        lens_config.left_xyz[ 0 ],
        lens_config.left_xyz[ 1 ],
        lens_config.left_xyz[ 2 ] );

    printf( "VR lens offset (right): (%f, %f, %f)\n",
        lens_config.right_xyz[ 0 ],
        lens_config.right_xyz[ 1 ],
        lens_config.right_xyz[ 2 ] );

    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_set_lens_configuration

Function

Sets the current lens configuration in the tracker.

Syntax

```
#include <tobii/tobii_wearable.h>
tobii_error_t TOBII_CALL tobii_set_lens_configuration( tobii_device_t* device,
  tobii_lens_configuration_t const* lens_config );
```

Remarks

device must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

lens_config must be a pointer to a valid tobii_lens_configuration_t. Upon success, the values have been written to the tracker. They should correspond to the physical attributes of the headset that they represent.

- *left* An array of three floats, for the x, y and z offset of the left lens in the headset, given in millimeters.
- *right* An array of three floats, for the x, y and z offset of the right lens in the headset, given in millimeters.

Return value

If the operation is successful, tobii_get_lens_configuration() returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_get_lens_configuration returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *lens_config* parameter was passed in as NULL.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

  The provided license does not permit this operation.

- **TOBII_ERROR_NOT_SUPPORTED**

  The device doesn't support this functionality. This error is returned if the API is called with a non-VR device.

- **TOBII_ERROR_CONNECTION_FAILED**

  The connection to the device was lost. Call tobii_device_reconnect() to re-establish connection.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_set_lens_configuration from within a callback function is not supported.

See also tobii_get_lens_configuration()

Example

```
#include <tobii/tobii_wearable.h>
#include <stdio.h>
#include <assert.h>

static void url_receiver( char const* url, void* user_data )
{
    char* buffer = (char*)user_data;
    if( *buffer != '\0' ) return; // only keep first value

    if( strlen( url ) < 256 )
        strcpy( buffer, url );
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    char url[ 256 ] = { 0 };
    error = tobii_enumerate_local_device_urls( api, url_receiver, url );
    assert( error == TOBII_ERROR_NO_ERROR && *url != '\0' );

    tobii_device_t* device;
    error = tobii_device_create( api, url, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_lens_configuration_writable_t writable;
    error = tobii_lens_configuration_writable( device, &writable );
    assert( error == TOBII_ERROR_NO_ERROR );

    if( writable == TOBII_LENS_CONFIGURATION_WRITABLE )
    {
        tobii_lens_configuration_t lens_config;
        //Add 32 mm offset for each lens on the X-axis
        lens_config.left_xyz[ 0 ] = 32.0;
        lens_config.right_xyz[ 0 ] = -32.0;

        lens_config.left_xyz[ 1 ] = 0.0;
        lens_config.right_xyz[ 1 ] = 0.0;

        lens_config.left_xyz[ 2 ] = 0.0;
        lens_config.right_xyz[ 2 ] = 0.0;

        error = tobii_set_lens_configuration( device, &lens_config );
        assert( error == TOBII_ERROR_NO_ERROR );
    }
    else
        printf( "Unable to write lens configuration to tracker\n" );


    error = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
```

```
}
```

tobii_lens_configuration_writable

Function Query the tracker whether it is possible to write a new lens configuration to it or not.

Syntax

```
#include <tobii/tobii_wearable.h>
tobii_error_t TOBII_CALL tobii_lens_configuration_writable( tobii_device_t* device,
    tobii_lens_configuration_writable_t* writable );
```

Remarks *device* must be a pointer to a valid tobii_device_t instance as created by calling tobii_device_create or tobii_device_create_ex.

writable must be a pointer to a valid tobii_lens_configuration_writable_t.

On success, *writable* will be assigned a value that tells whether the tracker can write a new lens configuration. **TOBII_LENS_CONFIGURATION_WRITABLE** if it is writable and **TOBII_LENS_CONFIGURATION_NOT_WRITABLE** if not.

Return value If the operation is successful, tobii_lens_configuration_writable() returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_lens_configuration_writable returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *writable* parameter was passed in as NULL.

- **TOBII_ERROR_CONNECTION_FAILED**

  The connection to the device was lost. Call tobii_device_reconnect() to re-establish connection.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

  The function failed because it was called from within a callback triggered from an API call such as tobii_device_process_callbacks(), tobii_calibration_retrieve(), tobii_enumerate_illumination_modes(), or tobii_license_key_retrieve(). Calling tobii_lens_configuration_writable from within a callback function is not supported.

See also tobii_get_lens_configuration(), tobii_set_lens_configuration()

tobii_engine.h

The tobii_engine.h header file is used for acquiring information about connected devices. It creates a dedicated connection to the tobii engine, which runs as a service/daemon depending on the operating system. Using this interface the user can subscribe to events about added and removed tracker devices, as well as changes in individual tracker readiness states. The readiness states gives the user information about which readiness state the tracker is currently in, i.e. if a firmware upgrade is in progress, a calibration is needed or if the tracker is ready for use.

Please note that there can only be one subscription callback registered to a stream at a time, i.e the tobii_device_list_change_subscribe. To register a new callback, first unsubscribe from the stream, then resubscribe with the new callback function.

Do NOT call StreamEngine API functions from within the callback functions, due to risk of internal deadlocks. Generally one should finish the callback functions as quickly as possible and not make any blocking calls.

tobii_engine_create

Function

Creates a device instance to be used for communicating with a specific device.

Syntax

```
#include <tobii/tobii_engine.h>
tobii_error_t TOBII_CALL tobii_engine_create( tobii_api_t* api, tobii_engine_t** engine );
```

Remarks

Stream engine establishes a communication channel to the engine and keeps track of internal states, tobii_engine_create allocates and initializes this state and establishes the connection. This connection can then be used to query the engine for more information about which trackers are connected to the system.

api must be a pointer to a valid tobii_api_t as created by calling tobii_api_create.

engine must be a pointer to a variable of the type tobii_engine_t* that is, a pointer to a tobii_engine_t-pointer. This variable will be filled in with a pointer to the created engine instance. tobii_engine_t is an opaque type.

Return value

If the engine is successfully created, tobii_engine_create returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_engine_create returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *api* or *engine* parameters were passed in as NULL.

- **TOBII_ERROR_ALLOCATION_FAILED**

  The internal call to malloc or to the custom memory allocator (if used) returned NULL, so engine creation failed.

- **TOBII_ERROR_NOT_AVAILABLE**

  A connection to the tobii engine could not be established. This error is returned if the tobii engine is not running, not installed on the system or if it is an older version of tobii engine.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

See also

tobii_engine_destroy(), tobii_api_create()

Example

```
#include <tobii/tobii.h>
#include <tobii/tobii_engine.h>
#include <stdio.h>
#include <assert.h>

void device_list_change_callback( char const* url, tobii_device_list_change_type_t type,
```

```
    tobii_device_readiness_t readiness, int64_t timestamp_us, void* user_data )
{
    (void)readiness; (void)timestamp_us; (void) user_data;
    switch( type )
    {
        case TOBII_DEVICE_LIST_CHANGE_TYPE_ADDED:
            printf( "A device with url:%s, has been added\n", url );
            break;
        case TOBII_DEVICE_LIST_CHANGE_TYPE_REMOVED:
            printf( "The device with url:%s, has been removed\n", url );
            break;
        case TOBII_DEVICE_LIST_CHANGE_TYPE_CHANGED:
            printf( "Readiness state changed for device with url:%s\n", url );
            break;
        default:
            printf( "Unknow device list change type\n" );
            break;
    }
}

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_engine_t* engine;
    error = tobii_engine_create( api, &engine );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_device_list_change_subscribe( engine, device_list_change_callback, 0 );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( engine, 0, NULL );
        assert( error == TOBII_ERROR_NO_ERROR || error == TOBII_ERROR_TIMED_OUT );

        error = tobii_engine_process_callbacks( engine );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_device_list_change_unsubscribe( engine );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_engine_destroy( engine );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );
    return 0;
}
```

tobii_engine_destroy

Function Destroy an engine previously created through a call to tobii_engine_create.

Syntax

```
#include <tobii/tobii_engine.h>
tobii_error_t tobii_engine_destroy( tobii_engine_t* engine );
```

Remarks tobii_engine_destroy disconnects from the engine, perform cleanup and free the memory allocated by calling tobii_engine_create.

NOTE: Make sure that no background thread is using the engine, for example in the thread calling tobii_engine_process_callbacks, before calling tobii_engine_destroy in order to avoid the risk of encountering undefined behavior.

engine must be a pointer to a valid tobii_engine_t instance as created by calling tobii_engine_create or tobii_engine_create_ex.

Return value If the engine was successfully destroyed, tobii_engine_destroy returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_engine_destroy returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *engine* parameter was passed in as NULL.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

See also tobii_engine_create()

Example See tobii_engine_create()

tobii_engine_reconnect

Function Establish a new connection after a disconnect.

Syntax

```
#include <tobii/tobii_engine.h>
tobii_error_t tobii_engine_reconnect( tobii_engine_t* engine );
```

Remarks When receiving the error code TOBII_ERROR_CONNECTION_FAILED, it is necessary to explicitly request reconnection, by calling tobii_engine_reconnect.

engine must be a pointer to a valid tobii_engine_t instance as created by calling tobii_engine_create.

Return value

- **TOBII_ERROR_INVALID_PARAMETER**

  The *engine* parameter was passed in as NULL.

- **TOBII_ERROR_CONNECTION_FAILED**

  When attempting to reconnect, a connection could not be established. You might want to wait for a bit and try again, for a few times, and if the problem persists, display a message for the user.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

See also tobii_engine_process_callbacks()

Example See tobii_engine_process_callbacks()

tobii_engine_process_callbacks

Function Receives data packages from the engine, and sends the data through any registered callbacks.

Syntax

```
#include <tobii/tobii_engine.h>
tobii_error_t tobii_engine_process_callbacks( tobii_engine_t* engine );
```

Remarks Stream engine does not do any kind of background processing, it doesn't start any threads. It doesn't use any asynchronous callbacks. This means that in order to receive data from the engine, the application needs to manually request the callbacks to happen synchronously, and this is done by calling tobii_engine_process_callbacks.

tobii_engine_process_callbacks will receive any data packages that are incoming from the engine, process them and call any subscribed callbacks with the data. No callbacks will be called outside of tobii_engine_process_callbacks, so the application have full control over when to receive callbacks.

tobii_engine_process_callbacks will not wait for data, and will early-out if there's nothing to process. In order to maintain the connection to the engine, tobii_engine_process_callbacks should be called at least 10 times per second.

The recommended way to use tobii_engine_process_callbacks, is to start a dedicated thread, and alternately call tobii_wait_for_callbacks and tobii_engine_process_callbacks. See tobii_wait_for_callbacks() for more details.

If there is already a suitable thread to regularly run tobii_engine_process_callbacks from (possibly interleaved with application specific operations), it is possible to do this without calling tobii_wait_for_callbacks(). In this scenario, time synchronization needs to be handled manually or the timestamps will start drifting. See tobii_update_timesync() for more details.

engine must be a pointer to a valid tobii_engine_t instance as created by calling tobii_engine_create.

Return value

If the operation is successful, tobii_engine_process_callbacks returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_engine_process_callbacks returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *engine* parameter was passed in as NULL.

- **TOBII_ERROR_ALLOCATION_FAILED**

  The internal call to malloc or to the custom memory allocator (if used) returned NULL, so engine creation failed.

- **TOBII_ERROR_CONNECTION_FAILED**

  The connection to the tobii engine was lost. Call tobii_engine_reconnect() to re-establish connection.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

See also

tobii_wait_for_callbacks(), tobii_engine_clear_callback_buffers(), tobii_engine_reconnect(), tobii_update_timesync()

Example

```
#include <tobii/tobii.h>
#include <tobii/tobii_engine.h>
#include <stdio.h>
#include <assert.h>

int main()
{
    tobii_api_t* api;
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_engine_t* engine;
    error = tobii_engine_create( api, &engine );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 1000; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        // other parts of main loop would be executed here

        error = tobii_engine_process_callbacks( engine );
        assert( error == TOBII_ERROR_NO_ERROR );
    }

    error = tobii_engine_destroy( engine );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );

    return 0;
}
```

tobii_engine_clear_callback_buffers

Function Removes all unprocessed entries from the callback queues.

Syntax

```
#include <tobii/tobii_engine.h>
tobii_error_t tobii_engine_clear_callback_buffers( tobii_engine_t* engine );
```

Remarks All the data that is received and processed are written into internal buffers used for the callbacks. In some circumstances, for example during initialization, you might want to discard any data that has been buffered but not processed, without having to destroy/recreate the engine, and without having to implement the filtering out of unwanted data. tobii_engine_clear_callback_buffers will clear all buffered data, and only data arriving *after* the call to tobii_engine_clear_callback_buffers will be forwarded to callbacks.

engine must be a pointer to a valid tobii_engine_t instance as created by calling tobii_engine_create.

Return value If the operation is successful, tobii_engine_clear_callback_buffers returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_engine_clear_callback_buffers returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *engine* parameter was passed in as NULL.

See also tobii_wait_for_callbacks(), tobii_engine_process_callbacks()

tobii_enumerate_devices

Function Retrieves information about trackers that are currently connected to the system.

Syntax

```
#include <tobii/tobii_engine.h>
tobii_error_t tobii_enumerate_devices( tobii_engine_t* engine,
    tobii_enumerated_device_receiver_t receiver, void* user_data );
```

Remarks A system might have multiple devices connected, which the stream engine is able to communicate with. tobii_enumerate_devices retrives a list of all such devices found and their respective readiness state. It will only list locally connected devices, not devices connected on the network.

engine must be a pointer to a valid tobii_engine_t instance as created by calling tobii_engine_create.

receiver is a function pointer to a function with the prototype:

```
void enumerated_device_receiver( tobii_enumerated_device_t const* enumerated_device, void*
user_data );
```

This function will be called for each device found during enumeration. It is called with the following parameters:

- *enumerated_devices* This is a pointer to a struct containing the following data:
  - *url* The URL for the device, zero terminated ASCII string.
  - *serial_number* The serial number of the device, zero terminated ASCII string.
  - *model* The model of the device, zero terminated ASCII string.
  - *generation* The generation of the device, zero terminated ASCII string.
  - *firmware_version* The firmware version of the device, zero terminated ASCII string.
  - *integration* The integration type of the device, zero terminated ASCII string.
  - *readiness* is one of the enum values in tobii_device_readiness_t:

- **TOBII_DEVICE_READINESS_WAITING_FOR_FIRMWARE_UPGRADE**

  Is the device waiting for a firmware upgrade.

- **TOBII_DEVICE_READINESS_UPGRADING_FIRMWARE**

  Is the device in the process of upgrading firmware.

- **TOBII_DEVICE_READINESS_WAITING_FOR_DISPLAY_AREA**

  Is the device waiting for display area information, this requires the client to set the display area.

- **TOBII_DEVICE_READINESS_WAITING_FOR_CALIBRATION**

  Is the device waiting for a valid calibration, this requires the client to perform or set a calibration.

- **TOBII_DEVICE_READINESS_CALIBRATING**

  Is the device in the process of calibrating.

- **TOBII_DEVICE_READINESS_READY**

  Is the device ready for use by the client.

- **TOBII_DEVICE_READINESS_PAUSED**

  Is the device in a paused state, the tracker is disabled while in the paused state.

- **TOBII_DEVICE_READINESS_MALFUNCTIONING**

  Is the device in a malfunctioning state, the tracker can not be used when in this state.

- *user_data* This is the custom pointer sent in to tobii_enumerate_devices.

user_data custom pointer which will be passed unmodified to the receiver function.

Return value

If the operation is successful, tobii_enumerate_devices returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_enumerate_devices returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *api* or *receiver* parameters has been passed in as NULL.

- **TOBII_ERROR_NOT_SUPPORTED**

  The tobii engine doesn't support the ability to list enumerated devices. This error is returned if the API is called with an old tobii engine.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.

See also

tobii_engine_create(), tobii_device_list_change_subscribe()

Example

```
#include <tobii/tobii.h>
#include <tobii/tobii_engine.h>
#include <stdio.h>
#include <assert.h>

void enumerated_device_receiver( tobii_enumerated_device_t const* enumerated_device, void*
user_data )
{
    int* count = (int*) user_data;
    ++(*count);
    printf( "%d. %s (%s)\n", *count, enumerated_device->model, enumerated_device->url );
}

int main()
{
    tobii_api_t* api;
```

```
    tobii_error_t error = tobii_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_engine_t* engine;
    error = tobii_engine_create( api, &engine );
    assert( error == TOBII_ERROR_NO_ERROR );

    int count = 0;
    error = tobii_enumerate_devices( engine, enumerated_device_receiver, &count );
    if( error == TOBII_ERROR_NO_ERROR )
        printf( "Found %d devices.\n", count );
    else
        printf( "Enumeration failed.\n" );

    error = tobii_engine_destroy( engine );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_api_destroy( api );
    assert( error == TOBII_ERROR_NO_ERROR );

    return 0;
}
```

tobii_device_list_change_subscribe

Function Start listening for connected device events.

Syntax

```
#include <tobii/tobii_engine.h>
tobii_error_t tobii_device_list_change_subscribe( tobii_engine_t* engine,
    tobii_device_list_change_callback_t callback, void* user_data );
```

Remarks This subscription is for receiving information regarding state changes for connected devices. A device list change event is triggered when a device has been added, removed or the readiness state of a device have changed.

engine must be a pointer to a valid tobii_engine_t instance as created by calling tobii_engine_create.

callback is a function pointer to a function with the prototype:

```
void tobii_device_list_change_callback( char const* url, tobii_device_list_change_type_t type,
    tobii_device_readiness_t readiness, int64_t timestamp_us, void* user_data );
```

This function will be called when there is a change to the device list. It is called with the following parameters:

- *url* The URL string for the device, zero terminated. This pointer will be invalid after returning from the function, so ensure you make a copy of the string rather than storing the pointer directly.
- *type* is one of the enum values in tobii_device_list_change_type_t:
  - **TOBII_DEVICE_LIST_CHANGE_TYPE_ADDED**

    The device has been added to the system.
  - **TOBII_DEVICE_LIST_CHANGE_TYPE_REMOVED**

    The device has been removed from the system.
  - **TOBII_DEVICE_LIST_CHANGE_TYPE_CHANGED**

    The device readiness state has changed.
- *readiness* is one of the enum values in tobii_device_readiness_t:
  - **TOBII_DEVICE_READINESS_WAITING_FOR_FIRMWARE_UPGRADE**

    Is the device waiting for a firmware upgrade.
  - **TOBII_DEVICE_READINESS_UPGRADING_FIRMWARE**

Is the device in the process of upgrading firmware.

- **TOBII_DEVICE_READINESS_WAITING_FOR_DISPLAY_AREA**

  Is the device waiting for display area information, this requires the client to set the display area.

- **TOBII_DEVICE_READINESS_WAITING_FOR_CALIBRATION**

  Is the device waiting for a valid calibration, this requires the client to perform or set a calibration.

- **TOBII_DEVICE_READINESS_CALIBRATING**

  Is the device in the process of calibrating.

- **TOBII_DEVICE_READINESS_READY**

  Is the device ready for use by the client.

- **TOBII_DEVICE_READINESS_PAUSED**

  Is the device in a paused state, the tracker is disabled while in the paused state.

- **TOBII_DEVICE_READINESS_MALFUNCTIONING**

  Is the device in a malfunctioning state, the tracker can not be used when in this state.

- *timestamp_us* Timestamp value for when the device list change event occurred, measured in microseconds (us). The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values. The function tobii_system_clock() can be used to retrieve a timestamp using the same clock and same relative values as this timestamp.
- *user_data* This is the custom pointer sent in when registering the callback.

user_data custom pointer which will be passed unmodified to the callback.

Return value

If the operation is successful, tobii_device_list_change_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_device_list_change_subscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *engine* or *callback* parameters were passed in as NULL.

- **TOBII_ERROR_ALREADY_SUBSCRIBED**

  A subscription for device list changes were already made. There can only be one callback registered at a time. To change to another callback, first call tobii_device_list_change_unsubscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The tobii engine doesn't support the stream. This error is returned if the API is called with an old tobii engine.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

See also

tobii_device_list_change_unsubscribe(), tobii_engine_process_callbacks(), tobii_system_clock()

Example

See tobii_engine_create()

tobii_device_list_change_unsubscribe

Function

Stops listening to gaze point stream that was subscribed to by a call to

tobii_gaze_point_subscribe()

Syntax

```
#include <tobii/tobii_engine.h>
tobii_error_t tobii_device_list_change_unsubscribe( tobii_engine_t* engine );
```

Remarks

engine must be a pointer to a valid tobii_engine_t instance as created by calling tobii_engine_create.

Return value

If the operation is successful, tobii_device_list_change_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_device_list_change_unsubscribe returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *engine* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUBSCRIBED**

  There was no subscription for device list changes. It is only valid to call tobii_device_list_change_unsubscribe() after first successfully calling tobii_device_list_change_subscribe().

- **TOBII_ERROR_NOT_SUPPORTED**

  The tobii engine doesn't support the stream. This error is returned if the API is called with an old tobii engine.

- **TOBII_ERROR_INTERNAL**

  Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support

See also

tobii_gaze_point_subscribe()

Example

See tobii_engine_create()