

## **Tobii Stream Engine API - Reference Documentation**

Tobii Eye Trackers generate eye tracking data streams (including user presence, headpose, etc) through the Tobii Stream Engine. Core functions and basic data streams are openly available through the Tobii Stream Engine API. Additional data streams, which provide more detailed eye tracking information and advanced functionality, are protected by license enforcement and are available for commercial licensing.

The Tobii Stream Engine API consists of the following modules.

- tobii - Core functions
- tobii\_streams - Basic gaze and eye tracking data streams
- tobii\_wearable - Basic gaze data streams for wearable VR devices
- tobii\_licensing - Functionality related to license enforcement mechanisms
- tobii\_config - Calibration and display setup (Only available through commercial licensing)
- tobii\_advanced - Advanced gaze data streams with detailed eye tracking information (Only available through commercial licensing)

The tobi.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

<b>Function</b>	Returns a printable error message.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; char const* tobii_error_message( tobii_error_t error );</pre>
<b>Remarks</b>	All other functions in the API returns an error code from the <code>tobii_error_t</code> enumeration. <code>tobii_error_message</code> translates from these error codes to a human readable message. If the value passed in the <i>error</i> parameter is not within the range of the <code>tobii_error_t</code> enum, a generic message is returned.
<b>Return value</b>	<code>tobii_error_message</code> returns a zero-terminated C string describing the specified error code. The string returned is statically allocated, so it should not be freed.
<b>Example</b>	<pre>#include &lt;tobii/tobii.h&gt; #include &lt;stdio.h&gt;  int main() {     tobii_api_t* api;      tobii_error_t error = tobii_api_create( &amp;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; }</pre>

## tobii\_get\_api\_version

<b>Function</b>	Query the current version of the API.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_get_api_version( tobii_version_t* version );</pre>
<b>Remarks</b>	<p><code>tobii_get_api_version</code> can be used to query the version of the stream engine dll currently used.</p> <p><i>version</i> is a pointer to an <code>tobii_version_t</code> variable to receive the current version numbers. It contains the following members:</p> <ul style="list-style-type: none"> <li>■ <i>major</i> incremented for API changes which are not backward-compatible.</li> <li>■ <i>minor</i> incremented for releases which add new, but backward-compatible, API features.</li> <li>■ <i>revision</i> incremented for minor changes and bug fixes which do not change the API.</li> <li>■ <i>build</i> incremented every time a new build is done, even when there are no changes.</li> </ul>
<b>Return value</b>	If the call is successful, <code>tobii_get_api_version</code> returns <code>TOBII_ERROR_NO_ERROR</code> . 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

<b>Function</b>	Destroys an API instance.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_api_destroy( tobii_api_t* api );</pre>
<b>Remarks</b>	<p>When creating an instance with <code>tobii_api_create</code>, some system resources are acquired. When finished using the API (typically during the shutdown process), <code>tobii_api_destroy</code> should be called to destroy the instance and ensure that those resources are released.</p> <p><code>tobii_api_destroy</code> should only be called if <code>tobii_api_create</code> completed successfully.</p> <p><i>api</i> must be a pointer to a valid <code>tobii_api_t</code> instance as created by calling <code>tobii_api_create</code>.</p>
<b>Return value</b>	<p>If the call was successful, <code>tobii_api_destroy</code> returns <b>TOBII_ERROR_NO_ERROR</b> otherwise it can return one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>api</i> parameter was passed in as NULL.</li> <li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.</li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_enumerate_illumination_modes()</code>, or <code>tobii_license_key_retrieve()</code>. Calling <code>tobii_api_destroy</code> from within a callback function is not supported.</li> </ul>
<b>See also</b>	<code>tobii_api_create()</code> , <code>tobii_device_destroy()</code>
<b>Example</b>	See <code>tobii_api_create()</code>

## tobii\_enumerate\_local\_device\_urls

<b>Function</b>	Retrieves the URLs for stream engine compatible devices currently connected to the system.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_enumerate_local_device_urls( tobii_api_t* api, tobii_device_url_receiver_t receiver, void* user_data );</pre>
<b>Remarks</b>	<p>A system might have multiple devices connected, which the stream engine is able to communicate with. <code>tobii_enumerate_local_device_urls</code> 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. Note that if both a</p>

tobii-ttp and a tobii-prp URL is available for the same tracker, only the tobii-prp URL will be reported. For details, see `tobii_enumerate_local_device_urls_ex()`.

*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

<b>Function</b>	Retrieves the URLs for the stream engine compatible devices, of the specified generation, currently connected to the system.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; 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 );</pre>
<b>Remarks</b>	A system might have multiple devices connected, which the stream engine is able to communicate with. <code>tobii_enumerate_local_device_urls_ex</code> works similar to <code>tobii_enumerate_local_device_urls()</code> , 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`

Note that PRP generation devices are always enumerated, and only the `tobii-prp` URL will be reported for a tracker for which there exists both a `tobii-ttp` and a `tobii-prp` URL.

#### 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_field_of_use_t field_of_use, 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.

User of the stream engine API needs to make a conscious decision regarding the intended field of use for the device by choosing between interactive or analytical use.

*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`.

*field\_of\_use* is one of the enum values in `tobii_field_of_use_t`:

### ■ TOBII\_FIELD\_OF\_USE\_INTERACTIVE

Device will be created for interactive use. No special license is required for this type use. Eye tracking data is only used as a user input for interaction experiences and cannot be stored, transmitted, nor analyzed or processed for other purposes.

### ■ TOBII\_FIELD\_OF\_USE\_ANALYTICAL

Device will be created for analytical use. This requires a special license from Tobii. Eye tracking data is used to analyze user attention, behavior or decisions in applications that store, transfer, record or analyze the data.

*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, the url string is not a valid device url (or NULL) or *tobii\_field\_of\_use\_t* value is not a valid value from `tobii_field_of_use_t` enum.

### ■ 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\_FIRMWARE\_UPGRADE\_IN\_PROGRESS

The firmware is currently in the process of being upgraded, try again in a little while.

### ■ 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, TOBII_FIELD_OF_USE_INTERACTIVE, &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

---

<b>Function</b>	Destroy a device previously created through a call to <code>tobii_device_create</code> .
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_device_destroy( tobii_device_t* device );</pre>
<b>Remarks</b>	<p><code>tobii_device_destroy</code> will disconnect from the device, perform cleanup and free the memory allocated by calling <code>tobii_device_create</code>.</p> <p><b>NOTE:</b> Make sure that no background thread is using the device, for example in the thread calling <code>tobii_device_process_callbacks</code>, before calling <code>tobii_device_destroy</code> in order to avoid the risk of encountering undefined behavior.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> or <code>tobii_device_create_ex</code>.</p>
<b>Return value</b>	<p>If the device is successfully destroyed, <code>tobii_device_create</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_device_create</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameter was passed in as NULL.</li> <li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.</li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_enumerate_illumination_modes()</code>, or <code>tobii_license_key_retrieve()</code>. Calling <code>tobii_device_destroy</code> from within a callback function is not supported.</li> </ul>
<b>See also</b>	<code>tobii_device_create()</code> , <code>tobii_device_create_ex()</code>
<b>Example</b>	See <code>tobii_device_create()</code>

## tobii\_wait\_for\_callbacks

---

<b>Function</b>	Puts the calling thread to sleep until there are new callbacks available to process.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_wait_for_callbacks( int device_count, tobii_device_t* const* devices )</pre>
<b>Remarks</b>	<p>Stream engine does not use any threads to do processing or receive data. Instead, the function <code>tobii_device_process_callbacks()</code> have to be called regularly, to receive data from the device, and process it.</p> <p>The typical use case is to implement your own thread to call <code>tobii_device_process_callbacks</code> from, and to avoid busy-waiting for data to become available, <code>tobii_wait_for_callbacks</code> can be called before each call to <code>tobii_device_process_callbacks</code>. It will sleep the calling thread until new data is available to process, after which</p>

tobii\_device\_process\_callbacks should be called to process it.

**Note:** tobii\_wait\_for\_callbacks() returning with **TOBII\_ERROR\_NO\_ERROR** does not necessarily indicate that one or more callbacks will be invoked by a subsequent call to tobii\_device\_process\_callbacks(), but rather that there is something to process.

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.

*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.

#### Return value

If the operation is successful, tobii\_wait\_for\_callbacks returns **TOBII\_ERROR\_NO\_ERROR**. If 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(), as it doesn't have any new data to process).

##### ■ TOBII\_ERROR\_INVALID\_PARAMETER

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

##### ■ TOBII\_ERROR\_CONFLICTING\_API\_INSTANCES

Every instance of device passed in must be created with the same instance of tobii\_api\_t. If different api instances were 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()

#### 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, TOBII_FIELD_OF_USE_INTERACTIVE, &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( 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;
}

```

## tobii\_device\_process\_callbacks

---

<b>Function</b>	Receives data packages from the device, and sends the data through any registered callbacks.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_device_process_callbacks( tobii_device_t* device );</pre>
<b>Remarks</b>	<p>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 <code>tobii_device_process_callbacks</code>.</p> <p><code>tobii_device_process_callbacks</code> 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 <code>tobii_device_process_callbacks</code>, so the application have full control over when to receive callbacks.</p> <p><code>tobii_device_process_callbacks</code> will not wait for data, and will early-out if there's nothing to process. In order to maintain the connection to the device, <code>tobii_device_process_callbacks</code> should be called at least 10 times per second.</p> <p>The recommended way to use <code>tobii_device_process_callbacks</code>, is to start a dedicated thread, and alternately call <code>tobii_wait_for_callbacks</code> and <code>tobii_device_process_callbacks</code>. See <code>tobii_wait_for_callbacks()</code> for more details.</p> <p>If there is already a suitable thread to regularly run <code>tobii_device_process_callbacks</code> from (possibly interleaved with application specific operations), it is possible to do this without calling <code>tobii_wait_for_callbacks()</code>. In this scenario, time synchronization needs to be handled manually or the timestamps will start drifting. See <code>tobii_update_timesync()</code> for more details.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> or <code>tobii_device_create_ex</code>.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_device_process_callbacks</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_device_process_callbacks</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameter was passed in as NULL.</li> <li>■ <b>TOBII_ERROR_ALLOCATION_FAILED</b> The internal call to malloc or to the custom memory allocator (if used) returned NULL, so device creation failed.</li> <li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</li> <li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.</li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_enumerate_illumination_modes()</code>, or <code>tobii_license_key_retrieve()</code>. Calling <code>tobii_device_process_callbacks</code> from within a callback function is not supported.</li> </ul>
<b>See also</b>	<code>tobii_wait_for_callbacks()</code> , <code>tobii_device_clear_callback_buffers()</code> , <code>tobii_device_reconnect()</code> , <code>tobii_update_timesync()</code>
<b>Example</b>	<pre>#include &lt;tobii/tobii.h&gt; #include &lt;stdio.h&gt; #include &lt;assert.h&gt;  static void url_receiver( char const* url, void* user_data ) {     char* buffer = (char*)user_data;</pre>

```

        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, TOBII_FIELD_OF_USE_INTERACTIVE, &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

---

<b>Function</b>	Removes all unprocessed entries from the callback queues.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_device_clear_callback_buffers( tobii_device_t* device );</pre>
<b>Remarks</b>	<p>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. <code>tobii_device_clear_callback_buffers</code> will clear all buffered data, and only data arriving <i>after</i> the call to <code>tobii_device_clear_callback_buffers</code> will be forwarded to callbacks.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> or <code>tobii_device_create_ex</code>.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_device_clear_callback_buffers</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_device_clear_callback_buffers</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> <p>The <i>device</i> parameter was passed in as NULL.</p> </li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> <p>The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_enumerate_illumination_modes()</code>, or <code>tobii_license_key_retrieve()</code>. Calling <code>tobii_device_clear_callback_buffers</code> from within a callback function is not supported.</p> </li> </ul>
<b>See also</b>	<code>tobii_wait_for_callbacks()</code> , <code>tobii_device_process_callbacks()</code>

## tobii\_device\_reconnect

---

<b>Function</b>	Establish a new connection after a disconnect.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt;</pre>

```
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_FIRMWARE_UPGRADE_IN_PROGRESS`

The firmware is currently in the process of being upgraded, try again in a little while.

##### ■ `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

Synchronizes the system clock with the device's hardware clock.

#### 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 synchronized. The system clock is used to generate timestamps for all streamed data and by `tobii_system_clock`. Only if either of these are of interest is it necessary to periodically synchronize, which is done by calling `tobii_update_timesync` every ~30 seconds.

This operation is in its nature unreliable and may be subject to packet loss.

*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.

- **TOBII\_ERROR\_CONNECTION\_FAILED** The connection to the device was lost. Call `tobii_device_reconnect()` to re-establish connection.
- **TOBII\_ERROR\_NOT\_SUPPORTED** The function failed because the operation is not supported by the connected tracker.

**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, TOBII_FIELD_OF_USE_INTERACTIVE, &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: %i 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, TOBII_FIELD_OF_USE_INTERACTIVE, &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

<b>Function</b>	Retrieves 3d coordinates of the track box frustum, given in millimeters from the device center.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_get_track_box( tobii_device_t* device, tobii_track_box_t* track_box );</pre>
<b>Remarks</b>	<p>The track box is a volume in front of the tracker within which the user can be tracked.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> or <code>tobii_device_create_ex</code>.</p> <p><i>track_box</i> is a pointer to a <code>tobii_track_box_t</code> variable to receive the result. It contains the following fields, all being arrays of three floating point values, describing the track box frustum:</p> <ul style="list-style-type: none"> <li>■ <i>front_upper_right_xyz, front_upper_left_xyz, front_lower_left_xyz, front_lower_right_xyz</i> The four points on the frustum plane closest to the device.</li> <li>■ <i>back_upper_right_xyz, back_upper_left_xyz, back_lower_left_xyz, back_lower_right_xyz</i> The four points on the frustum plane furthest from the device.</li> </ul>
<b>Return value</b>	<p>If track box coordinates were successfully retrieved, <code>tobii_get_track_box</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_get_track_box</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> One or more of the <i>device</i> and <i>track_box</i> parameters were passed in as NULL.</li> <li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</li> <li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.</li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_enumerate_illumination_modes()</code>, or <code>tobii_license_key_retrieve()</code>. Calling <code>tobii_get_track_box</code> from within a callback function is not supported.</li> </ul>
<b>Example</b>	<pre>#include &lt;tobii/tobii.h&gt; #include &lt;stdio.h&gt; #include &lt;assert.h&gt;</pre>



```

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, TOBII_FIELD_OF_USE_INTERACTIVE, &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, TOBII_FIELD_OF_USE_INTERACTIVE, &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

---

<b>Function</b>	Gets the current value of a state in the tracker.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_get_state_uint32( tobii_device_t* device, tobii_state_t state, uint32_t* value );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> 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, TOBII_FIELD_OF_USE_INTERACTIVE, &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 occurred 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 occurred 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, TOBII_FIELD_OF_USE_INTERACTIVE, &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

---

<b>Function</b>	Ask if a specific feature is supported or not.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_capability_supported( tobii_device_t* device,     tobii_capability_t capability, tobii_supported_t* supported );</pre>
<b>Remarks</b>	<p><i>device</i> must be a pointer to a valid <i>tobii_device_t</i> instance as created by calling <i>tobii_device_create</i> or <i>tobii_device_create_ex</i>.</p> <p><i>capability</i> is one of the enum values in <i>tobii_capability_t</i>:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_CAPABILITY_DISPLAY_AREA_WRITABLE</b> Query if the display area of the display can be changed by calling <i>tobii_set_display_area()</i>.</li> <li>■ <b>TOBII_CAPABILITY_CALIBRATION_2D</b> Query if the device supports performing 2D calibration by calling <i>tobii_calibration_collect_data_2d()</i>.</li> <li>■ <b>TOBII_CAPABILITY_CALIBRATION_3D</b> Query if the device supports performing 3D calibration by calling <i>tobii_calibration_collect_data_3d()</i>.</li> <li>■ <b>TOBII_CAPABILITY_PERSISTENT_STORAGE</b> Query if the device supports persistent storage, needed to use <i>tobii_license_key_store</i> and <i>tobii_license_key_retrieve</i>.</li> <li>■ <b>TOBII_CAPABILITY_CALIBRATION_PER_EYE</b> Query if the device supports per-eye calibration, needed to use the per-eye calibration api.</li> <li>■ <b>TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_3D_GAZE_COMBINED</b> Query if the device supports combined gaze point in the wearable data stream.</li> <li>■ <b>TOBII_CAPABILITY_FACE_TYPE</b> Query if the device supports face type setting, needed to use <i>tobii_get_face_type()</i>, <i>tobii_set_face_type()</i> and <i>tobii_enumerate_face_types()</i>.</li> <li>■ <b>TOBII_CAPABILITY_COMPOUND_STREAM_USER_POSITION_GUIDE_XY</b> Query if the device supports the x- and y-coordinates of the user position guide stream.</li> <li>■ <b>TOBII_CAPABILITY_COMPOUND_STREAM_USER_POSITION_GUIDE_Z</b> Query if the device supports the z-coordinate of the user position guide stream.</li> <li>■ <b>TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_LIMITED_IMAGE</b> Query if the device supports the wearable limited image stream.</li> <li>■ <b>TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_PUPIL_DIAMETER</b> Query if the device supports pupil diameter in the wearable data stream.</li> <li>■ <b>TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_PUPIL_POSITION</b> Query if the device supports pupil position in the wearable data stream.</li> <li>■ <b>TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_EYE_OPENNESS</b> Query if the device supports eye openness signal in the wearable data stream.</li> <li>■ <b>TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_3D_GAZE_PER_EYE</b> Query if the device supports per eye 3D gaze in the wearable data stream.</li> <li>■ <b>TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_USER_POSITION_GUIDE_XY</b></li> </ul>

Query if the device supports x- and y- coordinates of user position guide signal in the wearable data stream.

■ **TOBII\_CAPABILITY\_COMPOUND\_STREAM\_WEARABLE\_TRACKING\_IMPROVEMENTS**

**DEPRECATED** See alternative capabilities *IMPROVE\_USER\_POSITION\_HMD* and *INCREASE\_EYE\_RELIEF*

Query if the device supports tracking improvements in the wearable data stream.

■ **TOBII\_CAPABILITY\_COMPOUND\_STREAM\_WEARABLE\_CONVERGENCE\_DISTANCE**

Query if the device supports convergence distance in the wearable data stream.

■ **TOBII\_CAPABILITY\_COMPOUND\_STREAM\_WEARABLE\_IMPROVE\_USER\_POSITION\_HMD**

Query if the device supports the improve user position hmd signal in the wearable data stream.

■ **TOBII\_CAPABILITY\_COMPOUND\_STREAM\_WEARABLE\_INCREASE\_EYE\_RELIEF**

Query if the device supports the increase eye relief signal in the wearable data stream.

*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, TOBII_FIELD_OF_USE_INTERACTIVE, &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\_SYNCPOINT**
- **TOBII\_STREAM\_DIAGNOSTICS\_IMAGE**
- **TOBII\_STREAM\_CUSTOM**

*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, TOBII_FIELD_OF_USE_INTERACTIVE, &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 tobi\_streams.h header file is used for managing data stream subscriptions. There are several types of data streams in the API, and tobi\_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 tobi_gaze_point_subscribe( tobi_device_t* device,
    tobi_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 tobi\_device\_t instance as created by calling tobi\_device\_create.

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

```
void gaze_point_callback( tobi_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 tobi\_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, tobi\_gaze\_point\_subscribe returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, tobi\_gaze\_point\_subscribe returns an error code specific to the device.

**See also** tobi\_gaze\_point\_unsubscribe(), tobi\_device\_process\_callbacks(), tobi\_system\_clock()

**Example**

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

void gaze_point_callback( tobi_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, TOBII_FIELD_OF_USE_INTERACTIVE, &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( 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

---

<b>Function</b>	Stops listening to gaze point stream that was subscribed to by a call to <code>tobii_gaze_point_subscribe()</code>
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t tobii_gaze_point_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> .
<b>Return value</b>	If the operation is successful, <code>tobii_gaze_point_unsubscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_gaze_point_unsubscribe</code> returns an error code specific to the device.
<b>See also</b>	<code>tobii_gaze_point_subscribe()</code>
<b>Example</b>	See <code>tobii_gaze_point_subscribe()</code>

## tobii\_gaze\_origin\_subscribe

---

<b>Function</b>	Start listening for gaze origin data. Gaze origin is a point on the users eye, reported in millimeters from the center of the display.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t tobii_gaze_origin_subscribe( tobii_device_t* device,     tobii_gaze_origin_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	<p>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.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>callback</i> is a function pointer to a function with the prototype:</p>

```
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 an error code specific to the device.

#### 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, TOBII_FIELD_OF_USE_INTERACTIVE, &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( 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

---

<b>Function</b>	Stops listening to gaze origin stream that was subscribed to by a call to <code>tobii_gaze_origin_subscribe()</code>
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t tobii_gaze_origin_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> .
<b>Return value</b>	If the operation is successful, <code>tobii_gaze_origin_unsubscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_gaze_origin_unsubscribe</code> returns an error code specific to the device.
<b>See also</b>	<code>tobii_gaze_origin_subscribe()</code>
<b>Example</b>	See <code>tobii_gaze_origin_subscribe()</code>

## tobii\_eye\_position\_normalized\_subscribe

---

<b>Function</b>	Start listening for normalized eye position data. Eye position is a point on the users eye, reported in normalized track box coordinates.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t tobii_eye_position_normalized_subscribe( tobii_device_t* device,     tobii_eye_position_normalized_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	<p>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.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>callback</i> is a function pointer to a function with the prototype:</p> <pre>void eye_position_normalized_callback( tobii_eye_position_normalized_t const* eye_position, void* user_data )</pre> <p>This function will be called when there is new normalized eye position data available. It is called with the following parameters:</p> <ul style="list-style-type: none"> <li>■ <i>eye_position</i> <p>This is a pointer to a struct containing the following data:</p> <ul style="list-style-type: none"> <li>■ <i>timestamp_us</i> <p>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 <code>tobii_system_clock()</code> can be used to retrieve a timestamp using the same clock and same relative values as this timestamp.</p> </li> <li>■ <i>left_validity</i></li> </ul> </li> </ul>

**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 an error code specific to the device.

**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, TOBII_FIELD_OF_USE_INTERACTIVE, &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( 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

---

<b>Function</b>	Stops listening to normalized eye position stream that was subscribed to by a call to <code>tobii_eye_position_normalized_subscribe()</code>
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t tobii_eye_position_normalized_unsubscribe(     tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> .
<b>Return value</b>	If the operation is successful, <code>tobii_eye_position_normalized_unsubscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_eye_position_normalized_unsubscribe</code> returns an error code specific to the device.
<b>See also</b>	<code>tobii_eye_position_normalized_subscribe()</code>
<b>Example</b>	See <code>tobii_eye_position_normalized_subscribe()</code>

## tobii\_user\_presence\_subscribe

---

<b>Function</b>	Start listening for user presence notifications, reporting whether there is a person in front of the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t tobii_user_presence_subscribe( tobii_device_t* device,     tobii_user_presence_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	<p>This subscription is for being notified when a user is detected by the device, and when a user is no longer detected.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>callback</i> is a function pointer to a function with the prototype:</p> <pre>void presence_callback( tobii_user_presence_status_t status, int64_t timestamp_us,     void* user_data )</pre> <p>This function will be called when there is a change in presence state. It is called with the following parameters:</p> <ul style="list-style-type: none"> <li>■ <i>status</i> One of the following values: <ul style="list-style-type: none"> <li>■ <b>TOBII_USER_PRESENCE_STATUS_UNKNOWN</b> if user presence could not be determined.</li> <li>■ <b>TOBII_USER_PRESENCE_STATUS_AWAY</b> if there is a user in front of the device.</li> <li>■ <b>TOBII_USER_PRESENCE_STATUS_PRESENT</b> if there is no user in front of the device.</li> </ul> </li> <li>■ <i>timestamp_us</i> 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 <code>tobii_system_clock()</code> can be used to retrieve a timestamp using the same clock and same relative values as this timestamp.</li> <li>■ <i>user_data</i> This is the custom pointer sent in when registering the callback.</li> </ul> <p><i>user_data</i> custom pointer which will be passed unmodified to the callback.</p>
<b>Return value</b>	If the operation is successful, <code>tobii_user_presence_subscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_user_presence_subscribe</code> returns an error code specific to the device.
<b>See also</b>	<code>tobii_user_presence_unsubscribe()</code> , <code>tobii_device_process_callbacks()</code> , <code>tobii_system_clock()</code>
<b>Example</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; #include &lt;stdio.h&gt; #include &lt;assert.h&gt;</pre>

```

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, TOBII_FIELD_OF_USE_INTERACTIVE, &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( 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

<b>Function</b>	Stops listening to presence stream that was subscribed to by a call to <code>tobii_user_presence_subscribe()</code> .
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t tobii_user_presence_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> .
<b>Return value</b>	If the operation is successful, <code>tobii_user_presence_unsubscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_user_presence_unsubscribe</code> returns an error code specific to the device.
<b>See also</b>	<code>tobii_user_presence_subscribe()</code>
<b>Example</b>	See <code>tobii_user_presence_subscribe()</code>

## 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 element 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 an error code specific to the device.

**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, TOBII_FIELD_OF_USE_INTERACTIVE, &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( 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

---

<b>Function</b>	Stops listening to the head pose stream that was subscribed to by a call to <code>tobii_head_pose_subscribe()</code> .
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t TOBII_CALL tobii_head_pose_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> .
<b>Return value</b>	If the operation is successful, <code>tobii_head_pose_unsubscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_head_pose_unsubscribe</code> returns an error code specific to the device.
<b>See also</b>	<code>tobii_head_pose_subscribe()</code>
<b>Example</b>	See <code>tobii_head_pose_subscribe()</code>

## tobii\_notifications\_subscribe

---

<b>Function</b>	Start listening to the notifications stream, which reports state changes for a device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t tobii_notifications_subscribe( tobii_device_t* device,     tobii_notifications_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	<p>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 occurred on the device, as an effect of another client performing some operation (such as starting a calibration, or changing the display area).</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p>

*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_COMBINED_GAZE_EYE_SELECTION_CHANGED
TOBII_NOTIFICATION_TYPE_CALIBRATION_ID_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 an error code specific to the device.

**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, TOBII_FIELD_OF_USE_INTERACTIVE, &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( 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

---

<b>Function</b>	Stops listening to notifications stream that was subscribed to by a call to <code>tobii_notifications_subscribe()</code>
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t tobii_notifications_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> .
<b>Return value</b>	If the operation is successful, <code>tobii_notifications_unsubscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_notifications_unsubscribe</code> returns an error code specific to the device.
<b>See also</b>	<code>tobii_notifications_subscribe()</code>
<b>Example</b>	See <code>tobii_notifications_subscribe()</code>

## tobii\_user\_position\_guide\_subscribe

---

<b>Function</b>	Start listening to the user position guide stream, which is used to help a user position their eyes in the track box correctly. TODO: More and more indepth description of the user position guide stream
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t TOBII_CALL tobii_user_position_guide_subscribe( tobii_device_t* device,     tobii_user_position_guide_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	<p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>callback</i> is a function pointer to a function with the prototype: <code>void user_position_guide_callback( tobii_user_position_guide_t const * user_position_guide, void* user_data );</code></p> <p>This function will be called when there is a new position guide package to be sent to the subscriber. It is called with the following parameters:</p> <ul style="list-style-type: none"> <li>■ <i>user_position_guide</i></li> <li>■ <i>timestamp_us</i></li> </ul>

Timestamp value for when the user position guide 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\_position\_validity*

Indicates the validity of the `left_position_xyz` field. **TOBII\_VALIDITY\_INVALID** if the field is not valid, **TOBII\_VALIDITY\_VALID** if it is.

- *left\_position\_normalized\_xyz*

An array of three floats, for the x, y and z coordinates TODO: Description needed

- *right\_position\_validity*

Indicates the validity of the `right_position_xyz` field. **TOBII\_VALIDITY\_INVALID** if the field is not valid, **TOBII\_VALIDITY\_VALID** if it is.

- *right\_position\_normalized\_xyz*

An array of three floats, for the x, y and z coordinates TODO: Description needed

- *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_user_position_guide_subscribe` returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_user_position_guide_subscribe` returns an error code specific to the device.

**See also** `tobii_user_position_guide_unsubscribe()`

**Example**

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

void user_position_guide_callback( tobii_user_position_guide_t const* position_guide, void* user_data )
{
    if( position_guide->left_position_validity == TOBII_VALIDITY_VALID )
        printf( "Left position: (%f, %f, %f)\n",
            position_guide->left_position_normalized_xyz[ 0 ],
            position_guide->left_position_normalized_xyz[ 1 ],
            position_guide->left_position_normalized_xyz[ 2 ] );

    if( position_guide->right_position_validity == TOBII_VALIDITY_VALID )
        printf( "Right position: (%f, %f, %f)\n",
            position_guide->right_position_normalized_xyz[ 0 ],
            position_guide->right_position_normalized_xyz[ 1 ],
            position_guide->right_position_normalized_xyz[ 2 ] );
}

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, TOBII_FIELD_OF_USE_INTERACTIVE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_user_position_guide_subscribe( device, user_position_guide_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( 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_position_guide_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\_position\_guide\_unsubscribe

---

<b>Function</b>	Stops listening to the user position guide that was subscribed to by a call to <code>tobii_user_position_guide_subscribe()</code> .
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_streams.h&gt; tobii_error_t TOBII_CALL tobii_user_position_guide_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> .
<b>Return value</b>	If the operation is successful, <code>tobii_user_position_guide_unsubscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_user_position_guide_unsubscribe</code> returns an error code specific to the device.
<b>See also</b>	<code>tobii_user_position_guide_subscribe()</code>
<b>Example</b>	See <code>tobii_user_position_guide_subscribe()</code>

# 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.

**NOTE:** Fields marked **(BETA)** may be removed or changed without prior notice.

## tobii\_wearable\_consumer\_data\_subscribe

---

<b>Function</b>	Start listening for eye tracking data from wearable device, such as VR headsets.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_wearable.h&gt; tobii_error_t TOBII_CALL tobii_wearable_consumer_data_subscribe( tobii_device_t* device,     tobii_wearable_consumer_data_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	<p>All coordinates are expressed in a right-handed Cartesian system with Z facing forward from the eye.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> or <code>tobii_device_create_ex</code>.</p> <p><i>callback</i> is a function pointer to a function with the prototype:</p> <pre>void wearable_callback( tobii_wearable_consumer_data_t const* data, void* user_data )</pre> <p>This function will be called when there is new data available. It is called with the following parameters:</p> <ul style="list-style-type: none"><li>■ <i>data</i> 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.<ul style="list-style-type: none"><li>■ <i>timestamp_us</i> Timestamp value for when the data was captured, measured in microseconds (us), and synchronized with the clock of the computer. The function <code>tobii_system_clock</code> 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.</li><li>■ <i>left</i> This is a struct containing the following data, related to the left eye:<ul style="list-style-type: none"><li>■ <i>pupil_position_in_sensor_area_validity</i> <b>TOBII_VALIDITY_INVALID</b> if <i>pupil_position_in_sensor_area_xy</i> is not valid for this frame, <b>TOBII_VALIDITY_VALID</b> if it is.</li><li>■ <i>pupil_position_in_sensor_area_xy</i> 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.</li><li>■ <i>position_guide_validity</i> <b>TOBII_VALIDITY_INVALID</b> if <i>position_guide_xy</i> is not valid for this frame, <b>TOBII_VALIDITY_VALID</b> if it is.</li><li>■ <i>position_guide_xy</i> An array of two floats, for the x and y normalized positions per eye. The position should be compensated with the offset between lens and camera optical axis. 0.5: is the optimal position 0.3-0.7: is when the position is ok and all gaze use cases are supported (green eyes in the position guide app) 0-0.3 and 0.7-1: is when the system might still output gaze but performance is degraded (yellow eyes) &lt;0 and &gt;1: is when any gaze values are not reliable. No gaze use cases are supported (red eyes)</li><li>■ <i>blink_validity</i> <b>TOBII_VALIDITY_INVALID</b> if <i>blink</i> for the eye is not valid for this frame, <b>TOBII_VALIDITY_VALID</b> if it is.</li><li>■ <i>blink</i> A bool that represents if the user's eye is open, <b>TOBII_STATE_BOOL_FALSE</b> means the eye is open and <b>TOBII_STATE_BOOL_TRUE</b> the eye is closed.</li></ul></li><li>■ <i>right</i> This is another instance of the same struct as in <i>left</i>, but which holds data related to the right eye of the user.</li><li>■ <i>gaze_origin_combined_validity</i> <b>TOBII_VALIDITY_INVALID</b> if <i>gaze_origin_combined_mm_xyz</i> is not valid for this frame, <b>TOBII_VALIDITY_VALID</b> if it is.</li></ul></li></ul>

This field will only be set if you have the capability

TOBII\_CAPABILITY\_COMPOUND\_STREAM\_WEARABLE\_3D\_GAZE\_COMBINED. 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\_COMPOUND\_STREAM\_WEARABLE\_3D\_GAZE\_COMBINED. 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\_COMPOUND\_STREAM\_WEARABLE\_3D\_GAZE\_COMBINED. 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\_COMPOUND\_STREAM\_WEARABLE\_3D\_GAZE\_COMBINED. See `tobii_capability_supported()`.

- *convergence\_distance\_validity* **TOBII\_VALIDITY\_INVALID** if *convergence\_distance\_mm* is not valid for this frame, **TOBII\_VALIDITY\_VALID** if it is.
- *convergence\_distance\_mm* convergence distance in mm. It is the distance from the midpoint between both left and right cornea position and the intersection point.
- *improve\_user\_position\_hmd* **TOBII\_STATE\_BOOL\_TRUE** if the user needs to adjust the position of the HMD, otherwise **TOBII\_STATE\_BOOL\_FALSE**.

This field will only be set if you have the capability

TOBII\_CAPABILITY\_COMPOUND\_STREAM\_WEARABLE\_IMPROVE\_USER\_POSITION\_HMD. See `tobii_capability_supported()`.

- *increase\_eye\_relief* **TOBII\_STATE\_BOOL\_TRUE** if the user need to increase eye relief, i.e increase the distance between the eyes and the HMD, otherwise **TOBII\_STATE\_BOOL\_FALSE**.

This field will only be set if you have the capability

TOBII\_CAPABILITY\_COMPOUND\_STREAM\_WEARABLE\_INCREASE\_EYE\_RELIEF. 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_consumer_data_subscribe()` returns

**TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_wearable_consumer_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_consumer_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_consumer_data_subscribe` from within a callback function is not supported.

**See also** `tobii_wearable_consumer_data_unsubscribe()`, `tobii_device_process_callbacks()`, `tobii_capability_supported()`

### Example

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

void wearable_callback( tobii_wearable_consumer_data_t const* wearable, void* user_data )
{
    if( wearable->gaze_direction_combined_validity )
    {
        printf( "Combined gaze direction: (%f, %f, %f)\n",
            wearable->gaze_direction_combined_normalized_xyz[ 0 ],
            wearable->gaze_direction_combined_normalized_xyz[ 1 ],
            wearable->gaze_direction_combined_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, TOBII_FIELD_OF_USE_INTERACTIVE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_wearable_consumer_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( 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_consumer_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\_consumer\_data\_unsubscribe

**Function** Stops listening to the wearable data stream that was subscribed to by a call to `tobii_wearable_consumer_data_subscribe()`.

**Syntax** `#include <tobii/tobii_wearable.h>`



	<pre>tobii_error_t TOBII_CALL tobii_wearable_consumer_data_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> or <code>tobii_device_create_ex</code> .
<b>Return value</b>	<p>If the operation is successful, <code>tobii_wearable_consumer_data_unsubscribe()</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_wearable_consumer_data_unsubscribe</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameter was passed in as NULL.</li> <li>■ <b>TOBII_ERROR_NOT_SUBSCRIBED</b> There was no subscription for wearable data. It is only valid to call <code>tobii_wearable_consumer_data_unsubscribe()</code> after first successfully calling <code>tobii_wearable_consumer_data_subscribe()</code>.</li> <li>■ <b>TOBII_ERROR_NOT_SUPPORTED</b> 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.</li> <li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_enumerate_illumination_modes()</code>, or <code>tobii_license_key_retrieve()</code>. Calling <code>tobii_wearable_consumer_data_unsubscribe</code> from within a callback function is not supported.</li> </ul>
<b>See also</b>	<code>tobii_wearable_consumer_data_subscribe()</code>

## tobii\_wearable\_advanced\_data\_subscribe

---

<b>Function</b>	Start listening for eye tracking data from wearable device, such as VR headsets.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_wearable.h&gt; tobii_error_t TOBII_CALL tobii_wearable_advanced_data_subscribe( tobii_device_t* device,     tobii_wearable_advanced_data_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	<p>All coordinates are expressed in a right-handed Cartesian system with Z facing forward from the eye.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> or <code>tobii_device_create_ex</code>.</p> <p><i>callback</i> is a function pointer to a function with the prototype:</p> <pre>void wearable_callback( tobii_wearable_advanced_data_t const* data, void* user_data )</pre> <p>This function will be called when there is new data available. It is called with the following parameters:</p> <ul style="list-style-type: none"> <li>■ <i>data</i> 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. <ul style="list-style-type: none"> <li>■ <i>timestamp_tracker_us</i> Timestamp value for when the gaze data was captured in microseconds (us). It is generated on the device responsible for capturing the data. <i>timestamp_system_us</i> is generated using this value. The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values.</li> <li>■ <i>timestamp_system_us</i> Timestamp value for when the data was captured, measured in microseconds (us), and synchronized with the clock of the computer. The function <code>tobii_system_clock</code> 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.</li> <li>■ <i>left</i> This is a struct containing the following data, related to the left eye: <ul style="list-style-type: none"> <li>■ <i>gaze_origin_validity</i> <b>TOBII_VALIDITY_INVALID</b> if <i>gaze_origin_mm_xyz</i> is not valid for this frame, <b>TOBII_VALIDITY_VALID</b> if it is.</li> </ul> </li> </ul> </li> </ul>

- *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.
- *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.
- *position\_guide\_validity* **TOBII\_VALIDITY\_INVALID** if *position\_guide\_xy* is not valid for this frame, **TOBII\_VALIDITY\_VALID** if it is.
- *position\_guide\_xy* An array of two floats, for the x and y normalized positions per eye. The position should be compensated with the offset between lens and camera optical axis. 0.5: is the optimal position 0.3-0.7: is when the position is ok and all gaze use cases are supported (green eyes in the position guide app) 0-0.3 and 0.7-1: is when the system might still output gaze but performance is degraded (yellow eyes) <0 and >1: is when any gaze values are not reliable. No gaze use cases are supported (red eyes)
- *blink\_validity* **TOBII\_VALIDITY\_INVALID** if *blink* for the eye is not valid for this frame, **TOBII\_VALIDITY\_VALID** if it is.
- *blink* A bool that represents if the user's eye is open, **TOBII\_STATE\_BOOL\_FALSE** means the eye is open and **TOBII\_STATE\_BOOL\_TRUE** the eye is closed.
- *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\_COMPOUND\_STREAM\_WEARABLE\_3D\_GAZE\_COMBINED**. 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\_COMPOUND\_STREAM\_WEARABLE\_3D\_GAZE\_COMBINED**. 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\_COMPOUND\_STREAM\_WEARABLE\_3D\_GAZE\_COMBINED**. 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\_COMPOUND\_STREAM\_WEARABLE\_3D\_GAZE\_COMBINED**. See `tobii_capability_supported()`.

- *convergence\_distance\_validity* **TOBII\_VALIDITY\_INVALID** if *convergence\_distance\_mm* is not valid for this frame, **TOBII\_VALIDITY\_VALID** if it is.
- *convergence\_distance\_mm* convergence distance in mm. It is the distance from the midpoint between both left and right cornea position and the intersection point.
- *improve\_user\_position\_hmd* **TOBII\_STATE\_BOOL\_TRUE** if the user needs to adjust the position of the HMD, otherwise **TOBII\_STATE\_BOOL\_FALSE**.

This field will only be set if you have the capability  
TOBII\_CAPABILITY\_COMPOUND\_STREAM\_WEARABLE\_IMPROVE\_USER\_POSITION\_HMD.  
See `tobii_capability_supported()`.

- *increase\_eye\_relief* **TOBII\_STATE\_BOOL\_TRUE** if the user need to increase eye relief, i.e increase the distance between the eyes and the HMD, otherwise **TOBII\_STATE\_BOOL\_FALSE**.

This field will only be set if you have the capability  
TOBII\_CAPABILITY\_COMPOUND\_STREAM\_WEARABLE\_INCREASE\_EYE\_RELIEF. 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_advanced_data_subscribe()` returns **TOBII\_ERROR\_NO\_ERROR**.  
If the call fails, `tobii_wearable_advanced_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_advanced_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_advanced_data_subscribe` from within a callback function is not supported.

## See also

`tobii_wearable_advanced_data_unsubscribe()`, `tobii_device_process_callbacks()`, `tobii_capability_supported()`

## Example

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

void wearable_callback( tobii_wearable_advanced_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, TOBII_FIELD_OF_USE_INTERACTIVE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_wearable_advanced_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( 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_advanced_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\_advanced\_data\_unsubscribe

<b>Function</b>	Stops listening to the wearable data stream that was subscribed to by a call to <code>tobii_wearable_advanced_data_subscribe()</code> .
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_wearable.h&gt; tobii_error_t TOBII_CALL tobii_wearable_advanced_data_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> or <code>tobii_device_create_ex</code> .
<b>Return value</b>	If the operation is successful, <code>tobii_wearable_advanced_data_unsubscribe()</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_wearable_advanced_data_unsubscribe</code> returns one of the following: <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameter was passed in as NULL.</li> <li>■ <b>TOBII_ERROR_NOT_SUBSCRIBED</b> There was no subscription for wearable data. It is only valid to call <code>tobii_wearable_advanced_data_unsubscribe()</code> after first successfully calling <code>tobii_wearable_advanced_data_subscribe()</code>.</li> </ul>

#### ■ 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_advanced_data_unsubscribe` from within a callback function is not supported.

**See also** `tobii_wearable_advanced_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>
#include <string.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, TOBII_FIELD_OF_USE_INTERACTIVE, &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

<b>Function</b>	Sets the current lens configuration in the tracker.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_wearable.h&gt; tobii_error_t TOBII_CALL tobii_set_lens_configuration( tobii_device_t* device,     tobii_lens_configuration_t const* lens_config );</pre>
<b>Remarks</b>	<p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> or <code>tobii_device_create_ex</code>.</p> <p><i>lens_config</i> must be a pointer to a valid <code>tobii_lens_configuration_t</code>. Upon success, the values have been written to the tracker. They should correspond to the physical attributes of the headset that they represent.</p> <ul style="list-style-type: none"> <li>■ <i>left</i> An array of three floats, for the x, y and z offset of the left lens in the headset, given in millimeters.</li> <li>■ <i>right</i> An array of three floats, for the x, y and z offset of the right lens in the headset, given in millimeters.</li> </ul>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_get_lens_configuration()</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_get_lens_configuration</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> <p>The <i>device</i> or <i>lens_config</i> parameter was passed in as NULL.</p> </li> <li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> <p>The provided license does not permit this operation.</p> </li> <li>■ <b>TOBII_ERROR_NOT_SUPPORTED</b> <p>The device doesn't support this functionality. This error is returned if the API is called with a non-VR device.</p> </li> <li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> <p>The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</p> </li> </ul>

## ■ 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>
#include <string.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, TOBII_FIELD_OF_USE_INTERACTIVE, &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

---

<b>Function</b>	Query the tracker whether it is possible to write a new lens configuration to it or not.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_wearable.h&gt; tobii_error_t TOBII_CALL tobii_lens_configuration_writable( tobii_device_t* device,</pre>

```
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\_wearable\_foveated\_gaze\_subscribe

---

#### Function

Start listening for wearable foveated gaze stream.

#### Syntax

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

#### Remarks

This subscription is for receiving Wearable foveated stream. The stream gives information of left, right and combined eye gaze direction.

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

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

```
void tobii_wearable_foveated_gaze_callback( tobii_wearable_foveated_gaze_t const* data, void* user_data )
```

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

- *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.
- *tracking\_state* should be one of the following values:
  - **TOBII\_WEARABLE\_FOVEATED\_TRACKING\_STATE\_TRACKING**
  - **TOBII\_WEARABLE\_FOVEATED\_TRACKING\_STATE\_EXTRAPOLATED**
  - **TOBII\_WEARABLE\_FOVEATED\_TRACKING\_STATE\_LAST\_KNOWN**
- *float gaze\_direction\_combined\_normalized\_xyz* is combined eye gaze direction 3d vector normalized.
- *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_foveated_gaze_subscribe` returns **TOBII\_ERROR\_NO\_ERROR**. If



the call fails, `tobii_wearable_foveated_gaze_subscribe` returns one of the following:

■ **TOBII\_ERROR\_INVALID\_PARAMETER**

One or more of the *device* and *callback* parameters were passed in as NULL. *device* must be a valid `tobii_device_t` pointer as created by `tobii_device_create`, and *callback* must be a valid pointer to a `tobii_wearable_foveated_gaze_callback_t` function.

■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not valid, or has been blacklisted.

■ **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_wearable_foveated_gaze_subscribe()`.

■ **TOBII\_ERROR\_CALLBACK\_IN\_PROGRESS**

This function is called from a tobii call back function which is not allowed.

■ **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_wearable_foveated_gaze_unsubscribe()`, `tobii_device_process_callbacks()`

**Example**

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

void wearable_foveated_gaze_callback( tobii_wearable_foveated_gaze_t const* data,
                                      void* user_data )
{
    (void)user_data;
    printf( "Wearable foveated gaze %" PRIu64 " ", data->timestamp_us );

    printf( "Combined gaze: (%2f,%2f,%2f)", data->gaze_direction_combined_normalized_xyz[0],
            data->gaze_direction_combined_normalized_xyz[1], data->gaze_direction_combined_normalized_xyz[2] );

    printf("\n");
}

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

    error = tobii_wearable_foveated_gaze_subscribe( device, wearable_foveated_gaze_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( 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_foveated_gaze_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;
}
```

<b>Function</b>	Stops listening to wearable foveated gaze stream that was subscribed to by a call to <code>tobii_wearable_foveated_gaze_unsubscribe()</code>
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_wearable.h&gt; tobii_error_t TOBII_CALL tobii_wearable_foveated_gaze_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> as created by calling <code>tobii_device_create</code> .
<b>Return value</b>	<p>If the operation is successful, <code>tobii_wearable_foveated_gaze_unsubscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_wearable_foveated_gaze_unsubscribe</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b></li> </ul> <p>The <i>device</i> parameter was passed in as NULL. <i>device</i> must be a valid <code>tobii_device_t</code> pointer as created by <code>tobii_device_create</code>.</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b></li> </ul> <p>The provided license was not valid, or has been blacklisted.</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_NOT_SUBSCRIBED</b></li> </ul> <p>There was no subscription for gaze points. It is only valid to call <code>tobii_wearable_foveated_gaze_unsubscribe()</code> after first successfully calling <code>tobii_wearable_foveated_gaze_subscribe()</code>.</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b></li> </ul> <p>This function is called from a <code>tobii</code> call back function which is not allowed.</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INTERNAL</b></li> </ul> <p>Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</p>
<b>See also</b>	<code>tobii_wearable_foveated_gaze_subscribe()</code>

# tobii\_licensing.h

The tobii\_licensing.h file is used to manage functionality restricted by license provided by stream engine. What functionality is made available by stream engine is controlled by license files generated by Tobii.

The different levels of licenses are (from lowest to highest):

- Consumer (default, no license file required)
- Config
- Professional

Functionality available on lower levels is also available on higher levels.

## tobii\_device\_create\_ex

<b>Function</b>	Creates a device instance to be used for communicating with a specific device with a certain license.
-----------------	---

<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; TOBII_API tobii_error_t TOBII_CALL tobii_device_create_ex( tobii_api_t* api, char const* url, tobii_field_of_use_t field_of_use,     tobii_license_key_t const* license_keys, int license_count, tobii_license_validation_result_t* license_results, tobii_device_t** device );</pre>
---------------	---

<b>Remarks</b>	In order to communicate with a specific device, stream engine needs to keep track of a lot of internal state. tobii_device_create_ex 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.
----------------	--

User of the stream engine API needs to make a conscious decision regarding the intended field of use for the device by choosing between interactive or analytical use.

tobii\_license\_key\_t is a basic structure that contains the license key and its size in bytes.

A license key is used for enabling extended functionality of the engine under certain conditions. Conditions may include time limit, tracker model, tracker serial number, application name and/or application signature. Every license key have one feature group which gives them a set of features. They may also include additional features that are not included in their feature group. The device created will have all the features that provided by the valid licences passed as argument. If there is no valid license, the feature group of the device will be consumer level.

Licenses are provided by Tobii AB.

*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.

*field\_of\_use* is one of the enum values in tobii\_field\_of\_use\_t:

- **TOBII\_FIELD\_OF\_USE\_INTERACTIVE**

Device will be created for interactive use. No special license is required for this type use. Eye tracking data is only used as a user input for interaction experiences and cannot be stored, transmitted, nor analyzed or processed for other purposes.

- **TOBII\_FIELD\_OF\_USE\_ANALYTICAL**

Device will be created for analytical use. This requires a special license from Tobii. Eye tracking data is used to analyze user attention, behavior or decisions in applications that store, transfer, record or analyze the data.

*license\_keys* should be provided. It is an array of valid license keys provided by Tobii. At least one license must be provided. Some API functions requires a different license than the basic consumer license:

*license\_results* is optional. It is an array for returning the results of the license validation for each license. It is advised the check *license\_results* in any case. All the error's is related with licensing will only return by this array.

- **Professional** tobii\_gaze\_data\_subscribe(), tobii\_gaze\_data\_unsubscribe(),  
tobii\_digital\_syncport\_subscribe() tobii\_digital\_syncport\_unsubscribe() tobii\_timesync()  
tobii\_set\_illumination\_mode()
- **Config or Professional** tobii\_calibration\_start() tobii\_calibration\_stop()  
tobii\_calibration\_collect\_data\_2d() tobii\_calibration\_discard\_data\_2d() tobii\_calibration\_clear()

tobii\_calibration\_compute\_and\_apply() tobii\_calibration\_retrieve() tobii\_calibration\_apply()  
tobii\_set\_display\_area() tobii\_set\_output\_frequency() tobii\_set\_device\_name()

- **Additional Features** tobii\_image\_subscribe()

*count* must be greater than zero. It is the number of license keys has provided.

*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. `tobii_device_t` is an opaque type, and only its declaration is available in the API, its definition is internal.

#### 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*, *url*, *device* or *license\_keys* parameters were passed in as NULL, *tobii\_field\_of\_use\_t* value is not a valid value from `tobii_field_of_use_t` enum or the *count* parameter is less or equal to zero.

- **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\_FIRMWARE\_UPGRADE\_IN\_PROGRESS**

The firmware is currently in the process of being upgraded, try again in a little while.

- **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_ex` from within a callback function is not supported.

#### License Errors

- **TOBII\_LICENSE\_VALIDATION\_RESULT\_OK**

The license that has been provided is valid.

- **TOBII\_LICENSE\_VALIDATION\_RESULT\_TAMPERED**

The license file has been tampered.

- **TOBII\_LICENSE\_VALIDATION\_RESULT\_INVALID\_APPLICATION\_SIGNATURE**

The signature of the application that runs the stream engine is not same with the signature in the license file.

- **TOBII\_LICENSE\_VALIDATION\_RESULT\_NONSIGNED\_APPLICATION**

The application that runs the stream engine has not been signed.

- **TOBII\_LICENSE\_VALIDATION\_RESULT\_EXPIRED**

The validity of the license has been expired.

- **TOBII\_LICENSE\_VALIDATION\_RESULT\_PREMATURE**

The license is not valid yet.

- **TOBII\_LICENSE\_VALIDATION\_RESULT\_INVALID\_PROCESS\_NAME**

The process name of the application that runs the stream engine is not included to the list of process names in the license file.

- **TOBII\_LICENSE\_VALIDATION\_RESULT\_INVALID\_SERIAL\_NUMBER**

The serial number of the current eye tracker is not included to the list of serial numbers in the license file.

- **TOBII\_LICENSE\_VALIDATION\_RESULT\_INVALID\_MODEL**

The model name of the current eye tracker is not included to the list of model names in the license file.

#### ■ TOBII\_LICENSE\_VALIDATION\_RESULT\_INVALID\_PLATFORM\_TYPE

The platform type of the current eye tracker is not included to the list of platform types in the license file.

#### See also

tobii\_device\_destroy(), tobii\_enumerate\_local\_device\_urls(), tobii\_api\_create(), tobii\_get\_device\_info(), tobii\_get\_feature\_group() tobii\_device\_create()

#### Example

```
#include "tobii/tobii.h"
#include "tobii/tobii_licensing.h"

#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <assert.h>

static size_t read_license_file( uint16_t* license )
{
    FILE *license_file = fopen( "se_license_key_sample", "rb" );

    if( !license_file )
    {
        printf( "License key could not be found!" );
        return 0;
    }

    fseek( license_file, 0, SEEK_END );
    long file_size = ftell( license_file );
    rewind( license_file );

    if( file_size <= 0 )
    {
        printf( "License file is empty!" );
        return 0;
    }

    if( license )
    {
        fread( license, sizeof( uint16_t ), file_size / sizeof( uint16_t ), license_file );
    }

    fclose( license_file );
    return ( size_t )file_size;
}

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 );

    size_t license_size = read_license_file( 0 );
    assert( license_size > 0 );

    uint16_t* license_key = ( uint16_t* )malloc( license_size );
    memset( license_key, 0, license_size );
    read_license_file( license_key );

    tobii_license_key_t license = { license_key, license_size };
    tobii_license_validation_result_t validation_result;

    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_ex( api, url, TOBII_FIELD_OF_USE_INTERACTIVE, &license, 1,
    &validation_result, &device );
    free( license_key );
    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\_license\_key\_store

---

<b>Function</b>	Stores the license key on the tracker
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_license_key_store( tobii_device_t* device, void* data, size_t size );</pre>
<b>Remarks</b>	<p>license key can be stored on the device. Only one key will be stored on the device so calling the API will overwrite the old key. If either data or size is passed as 0 then it will erase the already stored license key.</p> <p><i>device</i> must be a pointer to a variable of the type <code>tobii_device_t*</code> that is, a pointer to a <code>tobii_device_t</code>.</p> <p><i>data</i> has to be in <code>uint16_t</code> text passed as the <code>void*</code>. It is optional and hence if it is 0 then it will erase already stored license</p> <p><i>size</i> is the no of bytes in the data buffer. If it is passed as 0 then it will erase already stored license.</p>
<b>Return value</b>	<p>If the device is successfully created, <code>tobii_device_create</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_device_create</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameter was passed in as NULL.</li> <li>■ <b>TOBII_ERROR_ALLOCATION_FAILED</b> The internal call to <code>malloc</code> or to the custom memory allocator (if used) returned NULL, so device creation failed.</li> <li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</li> <li>■ <b>TOBII_ERROR_NOT_SUPPORTED</b> The device doesn't support storage APIs. This error is returned if the API is called with an old device which doesn't support the license device store.</li> <li>■ <b>TOBII_ERROR_OPERATION_FAILED</b> Writing to the the device failed because of unexpected IO error, file not found, storage is full or filename is invalid.</li> <li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.</li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_enumerate_illumination_modes()</code>, or <code>tobii_license_key_retrieve()</code>. Calling <code>tobii_license_key_store</code> from within a callback function is not supported.</li> </ul>
<b>See also</b>	<code>tobii_license_key_retrieve()</code> , <code>tobii_device_create()</code>
<b>Example</b>	<pre>#include "tobii/tobii_licensing.h"  #include &lt;stdio.h&gt; #include &lt;malloc.h&gt; #include &lt;memory.h&gt; #include &lt;assert.h&gt;  static size_t read_license_file( uint16_t* license ) {     FILE *license_file = fopen( "se_license_key_sample", "rb" );      if( !license_file )     {         printf( "License key could not be found!" );     } }</pre>

```

        return 0;
    }

    fseek( license_file, 0, SEEK_END );
    long file_size = ftell( license_file );
    rewind( license_file );

    if( file_size <= 0 )
    {
        printf( "License file is empty!" );
        return 0;
    }

    if( license )
    {
        fread( license, sizeof( uint16_t ), file_size / sizeof( uint16_t ), license_file );
    }

    fclose( license_file );
    return ( size_t )file_size;
}

void data_receiver( void const* data, size_t size, void* user_data )
{
    if ( !data || !size || !user_data ) return; // user_data shouldn't be NULL if passed as Non NULL

    // The license is received here,
    // --> code to use the device would go here <--
    // We will just compare if the store was ok for demo pupose.
    tobii_license_key_t* license = ( tobii_license_key_t* )user_data;
    if( size != license->size_in_bytes ) return;
    if( !memcmp( (void*)license->license_key, data, size ) )
        printf("Data Received correctly");
    else
        printf( "Invalid Data Received" );
}

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 );

    size_t license_size = read_license_file( 0 );
    assert( license_size > 0 );

    uint16_t* license_key = ( uint16_t* )malloc( license_size );
    memset( license_key, 0, license_size );
    read_license_file( license_key );

    tobii_license_key_t license = { license_key, license_size };
    tobii_license_validation_result_t validation_result;

    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_ex( api, url, TOBII_FIELD_OF_USE_INTERACTIVE, &license, 1,
    &validation_result, &device );
    if ( error != TOBII_ERROR_NO_ERROR ) free( license_key );
    assert( error == TOBII_ERROR_NO_ERROR );

    // Store The license to the device
    error = tobii_license_key_store( device, (void*) license.license_key,
        license.size_in_bytes );
    if( error != TOBII_ERROR_NO_ERROR ) free( license_key );
    assert( error == TOBII_ERROR_NO_ERROR );

    // Retrieve the license from the device
    error = tobii_license_key_retrieve( device, data_receiver, (void*)&license );
    free( license_key );
    assert( error == TOBII_ERROR_NO_ERROR );

    // Erase the license from the device
    error = tobii_license_key_store( device, 0, 0 );
    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\_license\_key\_retrieve

---

<b>Function</b>	Retrieves the already stored license key from the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii.h&gt; tobii_error_t tobii_license_key_retrieve( tobii_device_t* device, tobii_data_receiver_t receiver, void* user_data );</pre>
<b>Remarks</b>	<p>The receiver function passed as the parameter receives the data. Instead of storing the pointer to data, content of the data should be copied as the data pointer becomes invalid after the call is over.</p> <p><i>device</i> must be a pointer to a variable of the type <code>tobii_device_t*</code> that is, a pointer to a <code>tobii_device_t</code>-pointer. the device is obtained by calling <code>tobii_device_create()</code> or by <code>tobii_device_create_ex()</code>. It must be freed by calling <code>tobii_device_destroy()</code> as clean up operation.</p> <p><i>receiver</i> is a function pointer to a function with the prototype:</p> <pre>void data_receiver( void const* data, size_t size, void* user_data )</pre> <p>This function will be called with the retrieved license data. It is called with the following parameters:</p> <ul style="list-style-type: none"> <li>■ <i>data</i> The license data read from device. This pointer will be invalid after returning from the function, so ensure you make a copy of the data rather than storing the pointer directly.</li> <li>■ <i>size</i> This gives the size of the data buffer read.</li> <li>■ <i>user_data</i> This is the custom pointer sent in to <code>tobii_license_key_retrieve</code>.</li> </ul> <p><i>user_data</i> is optional. Caller can pass any data here as the calling device which could be used in the <i>receiver</i>.</p>
<b>Return value</b>	<p>If the device is successfully created, <code>tobii_device_create</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_device_create</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameter was passed in as NULL.</li> <li>■ <b>TOBII_ERROR_ALLOCATION_FAILED</b> The internal call to malloc or to the custom memory allocator (if used) returned NULL, so device creation failed.</li> <li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</li> <li>■ <b>TOBII_ERROR_NOT_SUPPORTED</b> The device doesn't support storage APIs. This error is returned if the API is called with an old device which doesn't support the license device store.</li> <li>■ <b>TOBII_ERROR_OPERATION_FAILED</b> Reading from the device failed because of unexpected IO error, file not found, filename is invalid.</li> <li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support.</li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_enumerate_illumination_modes()</code>, or <code>tobii_license_key_retrieve()</code>. Calling <code>tobii_license_key_retrieve</code> from within a callback function is not supported.</li> </ul>
<b>See also</b>	<code>tobii_license_key_retrieve()</code> , <code>tobii_device_create()</code>



## Example

```
#include "tobii/tobii_licensing.h"

#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <assert.h>

static size_t read_license_file( uint16_t* license )
{
    FILE *license_file = fopen( "se_license_key_sample", "rb" );

    if( !license_file )
    {
        printf( "License key could not be found!" );
        return 0;
    }

    fseek( license_file, 0, SEEK_END );
    long file_size = ftell( license_file );
    rewind( license_file );

    if( file_size <= 0 )
    {
        printf( "License file is empty!" );
        return 0;
    }

    if( license )
    {
        fread( license, sizeof( uint16_t ), file_size / sizeof( uint16_t ), license_file );
    }

    fclose( license_file );
    return ( size_t )file_size;
}

void data_receiver( void const* data, size_t size, void* user_data )
{
    if ( !data || !size || !user_data ) return; // user_data shouldn't be NULL if passed as Non NULL

    // The license is received here,
    // --> code to use the device would go here <--
    // We will just compare if the store was ok for demo pupose.
    tobii_license_key_t* license = ( tobii_license_key_t* )user_data;
    if( size != license->size_in_bytes ) return;
    if( !memcmp( (void*)license->license_key, data, size ) )
        printf("Data Received correctly");
    else
        printf( "Invalid Data Received" );
}

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 );

    size_t license_size = read_license_file( 0 );
    assert( license_size > 0 );

    uint16_t* license_key = ( uint16_t* )malloc( license_size );
    memset( license_key, 0, license_size );
    read_license_file( license_key );

    tobii_license_key_t license = { license_key, license_size };
    tobii_license_validation_result_t validation_result;

    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_ex( api, url, TOBII_FIELD_OF_USE_INTERACTIVE, &license, 1,
    &validation_result, &device );
    if ( error != TOBII_ERROR_NO_ERROR ) free( license_key );
    assert( error == TOBII_ERROR_NO_ERROR );
}
```

```

// Store The license to the device
error = tobii_license_key_store( device, (void*) license.license_key,
    license.size in bytes );
if( error != TOBII_ERROR_NO_ERROR ) free( license_key );
assert( error == TOBII_ERROR_NO_ERROR );

// Retrieve the license from the device
error = tobii_license_key_retrieve( device, data_receiver, (void*)&license );
free( license_key );
assert( error == TOBII_ERROR_NO_ERROR );

// Erase the license from the device
error = tobii_license_key_store( device, 0, 0 );
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\_get\_feature\_group

<b>Function</b>	Retrieves the currently active feature group for a device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_advanced.h&gt; tobii_error_t tobii_get_feature_group( tobii_device_t* device, tobii_feature_group_t* feature_group );</pre>
<b>Remarks</b>	<p>The currently active feature group is determined by <code>tobii_device_create</code> based on the license key passed into it. <code>tobii_get_feature_group</code> can be used to query the currently active feature group.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> as created by calling <code>tobii_device_create</code>.</p> <p><i>feature_group</i> is a pointer to a <code>tobii_feature_group_t</code> to receive the current group, in the form of values from the following enum:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_FEATURE_GROUP_BLOCKED</b> <p>The provided license key was invalid, or the application making the call has been blacklisted. No API functionality will be available.</p> </li> <li>■ <b>TOBII_FEATURE_GROUP_CONSUMER</b> <p>Default feature group for passing a NULL (default) license key to <code>tobii_device_create</code>. Gives access to all API functions except those where a higher feature group is specified in the documentation.</p> </li> <li>■ <b>TOBII_FEATURE_GROUP_CONFIG</b> <p>Grants access to functionality that changes configuration of the tracker (mainly in <code>tobii_config.h</code>). This feature group might be automatically granted for certain devices, like head-mounted displays, even if a default license key is used.</p> </li> <li>■ <b>TOBII_FEATURE_GROUP_PROFESSIONAL</b> <p>Gives access to the functionality in <code>tobii_advanced.h</code>. This feature group might be automatically granted for professional level devices, as supplied by Tobii Pro, even if a default license key is used.</p> </li> <li>■ <b>TOBII_FEATURE_GROUP_INTERNAL</b> <p>For internal use by Tobii.</p> </li> </ul> <p>The current feature group controls which API features are available. The documentation will state which functions require a specific license (if it is not specified, it is assumed that <b>TOBII_FEATURE_GROUP_CONSUMER</b> is required).</p> <p>Each feature group includes all feature groups preceding it (with the exception of <b>TOBII_FEATURE_GROUP_BLOCKED</b>, which indicates that the specified license key was found to be invalid, or the current application has been blacklisted, in which case no API functions will be available).</p>
<b>Return value</b>	<p>If the feature group was successfully retrieved, <code>tobii_get_feature_group</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_get_feature_group</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> <p>One or more of the <i>device</i> and <i>feature_group</i> parameters were passed in as NULL. <i>device</i> must be a valid</p> </li> </ul>

tobii\_device\_t pointer as created by tobi\_device\_create, and *feature\_group* must be a valid pointer to a tobi\_feature\_group\_t variable.

#### ■ 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 tobi\_device\_process\_callbacks(), tobi\_calibration\_retrieve(), tobi\_enumerate\_illumination\_modes(), or tobi\_license\_key\_retrieve(). Calling tobi\_get\_feature\_group from within a callback function is not supported.

**See also** tobi\_device\_create()

#### Example

```
#include <tobii/tobii_licensing.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()
{
    tobi_api_t* api;
    tobi_error_t error = tobi_api_create( &api, NULL, NULL );
    assert( error == TOBII_ERROR_NO_ERROR );

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

    tobi_device_t* device;
    error = tobi_device_create( api, url, TOBII_FIELD_OF_USE_INTERACTIVE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobi_feature_group_t feature_group;
    error = tobi_get_feature_group( device, &feature_group );
    assert( error == TOBII_ERROR_NO_ERROR );

    if( feature_group == TOBII_FEATURE_GROUP_CONSUMER )
        printf( "Running with 'consumer' feature group.\n" );

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

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

# tobii\_config.h

The tobi\_config.h header file contains functionality to configure the tracker, such as calibration and display area setup. Note that using the configuration APIs incorrectly may cause some tracker functionality to work incorrectly. Please refer to the calibration sample for recommendations on how to implement a correct calibration.

All functions in the configuration API which modify state (i.e. everything except get- and enumerate-functions) require a license on at least config level, and a device created through tobi\_device\_create\_ex.

## tobii\_set\_enabled\_eye

---

**Function** Set the enabled eye prior to calibrating.

**Syntax**

```
#include <tobii/tobii_config.h>
tobii_error_t tobi_set_enabled_eye( tobi_device_t* device, tobi_enabled_eye_t enabled_eye );
```

**Remarks** *device* must be a pointer to a valid tobi\_device\_t instance as created by calling tobi\_device\_create\_ex with a valid config level license.

*enabled\_eye* contains the value to which the enabled eye property of the device shall be set: :  
**TOBII\_ENABLED\_EYE\_LEFT**, **TOBII\_ENABLED\_EYE\_RIGHT** or **TOBII\_ENABLED\_EYE\_BOTH**

**Return value** If the operation is successful, tobi\_set\_enabled\_eye returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, tobi\_set\_enabled\_eye returns one of the following:

- **TOBII\_ERROR\_INVALID\_PARAMETER**

The *device* parameter passed was NULL.

- **TOBII\_ERROR\_CALLBACK\_IN\_PROGRESS**

The function failed because it was called from within a callback triggered from an API call such as tobi\_device\_process\_callbacks(), tobi\_calibration\_retrieve() or tobi\_enumerate\_illumination\_modes(). Calling tobi\_get\_output\_frequency from within a callback function is not supported.

- **TOBII\_ERROR\_NOT\_SUPPORTED**

Setting the enabled eye property is not supported by the device.

- **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid license, or has been blacklisted.

- **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** tobi\_get\_enabled\_eye()

**Example**

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

int main()
{
    // See tobi.h for examples on how to create/destroy an api using tobi_api_create/destroy(),
    // tobi_licensing.h on how to create a device using tobi_device_create_ex()
    // and tobi.h on how to destroy a device using tobi_device_destroy().
    tobi_device_t* device = 0;

    tobi_enabled_eye_t enabled_eye = TOBII_ENABLED_EYE_BOTH;
    tobi_error_t error = tobi_get_enabled_eye( device, &enabled_eye );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Failed to get enabled eye property" );
    }

    switch( enabled_eye )
    {
        case TOBII_ENABLED_EYE_LEFT: printf( "TOBII_ENABLED_EYE_LEFT" ); break;
        case TOBII_ENABLED_EYE_RIGHT: printf( "TOBII_ENABLED_EYE_RIGHT" ); break;
        case TOBII_ENABLED_EYE_BOTH: printf( "TOBII_ENABLED_EYE_BOTH" ); break;
    }
```

```

    }
    return error;
}

```

## tobii\_get\_enabled\_eye

---

**Function** Queries the enabled eye property of the device.

**Syntax** `#include <tobii/tobii_config.h>`  
`tobii_error_t tobii_get_enabled_eye( tobii_device_t* device, tobii_enabled_eye_t* enabled_eye );`

**Remarks** TBD - Documentation needs to be written for this function

*device* must be a pointer to a valid `tobii_device_t` instance as created by calling `tobii_device_create_ex` with a valid config level license.

*enabled\_eye* is a valid pointer to the value containing the retrieved enabled eye property of the device:

**TOBII\_ENABLED\_EYE\_LEFT**, **TOBII\_ENABLED\_EYE\_RIGHT** or **TOBII\_ENABLED\_EYE\_BOTH**.

**Return value** If the operation is successful, `tobii_get_output_frequency` returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_get_enabled_eye` returns one of the following:

- **TOBII\_ERROR\_INVALID\_PARAMETER**

The *device* or *enabled\_eye* parameters were 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()` or `tobii_enumerate_illumination_modes()`. Calling `tobii_get_enabled_eye` from within a callback function is not supported.

- **TOBII\_ERROR\_NOT\_SUPPORTED**

Getting the enabled eye property is not supported by the device.

- **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid license, or has been blacklisted.

- **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_set_enabled_eye()`

**Example**

```

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

int main()
{
    // See tobii.h for examples on how to create/destroy an api using tobii_api_create/destroy(),
    // tobii_licensing.h on how to create a device using tobii_device_create_ex()
    // and tobii.h on how to destroy a device using tobii_device_destroy().
    tobii_device_t* device = 0;

    tobii_enabled_eye_t enabled_eye_props[] = { TOBII_ENABLED_EYE_BOTH, TOBII_ENABLED_EYE_LEFT,
    TOBII_ENABLED_EYE_RIGHT };

    //size_t ix = sizeof( enabled_eye_props ) / sizeof( enabled_eye_props[ 0 ] );
    for( size_t ix = 0; ix < sizeof( enabled_eye_props ) / sizeof( enabled_eye_props[ 0 ] ); ++ix )
    {
        tobii_error_t error = tobii_set_enabled_eye( device, enabled_eye_props[ ix ] );
        if( error != TOBII_ERROR_NO_ERROR )
        {
            printf( "Failure " );
        }
        else
        {
            printf( "Success " );
        }
    }

    printf( "setting enabled eye property to: " );
    switch( enabled_eye_props[ ix ] )
    {
        case TOBII_ENABLED_EYE_LEFT: printf( "TOBII_ENALBED_EYE_LEFT" );

```

```

        break;
    case TOBII_ENABLED_EYE_RIGHT: printf( "TOBII_ENALBED_EYE_RIGHT" );
        break;
    case TOBII_ENABLED_EYE_BOTH: printf( "TOBII_ENALBED_EYE_BOTH" );
        break;
    }
    printf( "\n" );
}
return 0;
}

```

## tobii\_calibration\_start

---

<b>Function</b>	Starts a calibration, placing the tracker in a state ready to receive data collection requests.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_calibration_start( tobii_device_t* device,     tobii_enabled_eye_t enabled_eye );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>enabled_eye</i> must ALWAYS be <b>TOBII_ENABLED_EYE_BOTH</b></p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_calibration_start</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_calibration_start</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> <p>The <i>device</i> parameter was passed in as NULL.</p> </li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> <p>The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_calibration_start</code> from within a callback function is not supported.</p> </li> <li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> <p>The provided license was not a valid config level license, or has been blacklisted.</p> </li> <li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> <p>The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</p> </li> <li>■ <b>TOBII_ERROR_CALIBRATION_ALREADY_STARTED</b> <p><code>tobii_calibration_start</code> has already been called, and not yet been stopped by calling <code>tobii_calibration_stop</code>. A started calibration must always be stopped before a new calibration is started.</p> </li> <li>■ <b>TOBII_ERROR_CALIBRATION_BUSY</b> <p>Another client is already calibrating the device. Only one calibration can be running at a time, across all connected clients.</p> </li> <li>■ <b>TOBII_ERROR_INTERNAL</b> <p>Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</p> </li> <li>■ <b>TOBII_ERROR_NOT_SUPPORTED</b> <p>A value other than <code>TOBII_ENABLED_EYE_BOTH</code> was passed for the <i>enabled_eye</i> parameter.</p> </li> </ul>
<b>See also</b>	<p><code>tobii_calibration_stop()</code>, <code>tobii_calibration_collect_data_2d()</code>, <code>tobii_calibration_collect_data_3d()</code>, <code>tobii_calibration_collect_data_per_eye_2d()</code>, <code>tobii_calibration_discard_data_2d()</code>, <code>tobii_calibration_discard_data_3d()</code>, <code>tobii_calibration_discard_data_per_eye_2d()</code>, <code>tobii_calibration_clear()</code>, <code>tobii_calibration_compute_and_apply()</code>, <code>tobii_calibration_compute_and_apply_per_eye()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_calibration_parse()</code>, <code>tobii_calibration_apply()</code></p>
<b>Example</b>	See <code>tobii_calibration_collect_data_2d()</code> .

## tobii\_calibration\_stop

<b>Function</b>	Signals that the calibration process has been completed, and that no further data collection will be requested.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_calibration_stop( tobii_device_t* device );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_calibration_stop</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_calibration_stop</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameter was passed in as NULL.</li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_calibration_stop</code> from within a callback function is not supported.</li> <li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> The provided license was not a valid config level license, or has been blacklisted.</li> <li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</li> <li>■ <b>TOBII_ERROR_CALIBRATION_NOT_STARTED</b> A successful call to <code>tobii_calibration_start</code> has not been made before calling this function.</li> <li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li> </ul>
<b>See also</b>	<code>tobii_calibration_start()</code> , <code>tobii_calibration_collect_data_2d()</code> , <code>tobii_calibration_collect_data_3d()</code> , <code>tobii_calibration_collect_data_per_eye_2d()</code> , <code>tobii_calibration_discard_data_2d()</code> , <code>tobii_calibration_discard_data_3d()</code> , <code>tobii_calibration_discard_data_per_eye_2d()</code> , <code>tobii_calibration_clear()</code> , <code>tobii_calibration_compute_and_apply()</code> , <code>tobii_calibration_compute_and_apply_per_eye()</code> , <code>tobii_calibration_retrieve()</code> , <code>tobii_calibration_parse()</code> , <code>tobii_calibration_apply()</code>
<b>Example</b>	See <code>tobii_calibration_collect_data_2d()</code> .

## tobii\_calibration\_collect\_data\_2d

<b>Function</b>	Performs data collection for the specified screen coordinate.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_calibration_collect_data_2d( tobii_device_t* device, float x, float y );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>x</i> the x-coordinate (horizontal) of the point to collect calibration data for, in normalized (0 to 1) coordinates.</p> <p><i>y</i> the y-coordinate (vertical) of the point to collect calibration data for, in normalized (0 to 1) coordinates.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_calibration_collect_data_2d</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_calibration_collect_data_2d</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameter was passed in as NULL.</li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_calibration_collect_data_2d</code> from within a callback function is not supported.</li> </ul>

- **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid config level license, or has been blacklisted.

- **TOBII\_ERROR\_CONNECTION\_FAILED**

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

- **TOBII\_ERROR\_CALIBRATION\_NOT\_STARTED**

A successful call to `tobii_calibration_start` has not been made before calling this function.

- **TOBII\_ERROR\_OPERATION\_FAILED**

The tracker failed to collect a sufficient amount of data. It is recommended to performing the operation again.

- **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_calibration_start()`, `tobii_calibration_stop()`, `tobii_calibration_collect_data_3d()`, `tobii_calibration_collect_data_per_eye_2d()`, `tobii_calibration_discard_data_2d()`, `tobii_calibration_discard_data_3d()`, `tobii_calibration_discard_data_per_eye_2d()`, `tobii_calibration_clear()`, `tobii_calibration_compute_and_apply()`, `tobii_calibration_compute_and_apply_per_eye()`, `tobii_calibration_retrieve()`, `tobii_calibration_parse()`, `tobii_calibration_apply()`

**Example**

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_calibration\_collect\_data\_3d

---

**Function**

Performs data collection for the specified 3d coordinate.

**Syntax**

```
#include <tobii/tobii_config.h>
tobii_error_t tobi_calibration_collect_data_3d( tobii_device_t* device,
    float x, float y, float z );
```

**Remarks**

TBD - Documentation needs to be written for this function

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

*x* the x-coordinate (horizontal) of the point to collect calibration data for, in millimeters.

*y* the y-coordinate (vertical) of the point to collect calibration data for, in millimeters.

*z* the z-coordinate (depth) of the point to collect calibration data for, in millimeters.

**Return value**

If the operation is successful, `tobii_calibration_collect_data_3d` returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_calibration_collect_data_3d` 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()` or `tobii_enumerate_illumination_modes()`. Calling `tobii_calibration_collect_data_3d` from within a callback function is not supported.

- **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid config level license, or has been blacklisted.

- **TOBII\_ERROR\_CONNECTION\_FAILED**

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

- **TOBII\_ERROR\_CALIBRATION\_NOT\_STARTED**

A successful call to `tobii_calibration_start` has not been made before calling this function.



#### ■ **TOBII\_ERROR\_OPERATION\_FAILED**

The tracker failed to collect a sufficient amount of data. It is recommended to perform the operation again.

#### ■ **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\_calibration\_start(), tobi\_calibration\_stop(), tobi\_calibration\_collect\_data\_2d(), tobi\_calibration\_collect\_data\_per\_eye\_2d(), tobi\_calibration\_discard\_data\_2d(), tobi\_calibration\_discard\_data\_3d(), tobi\_calibration\_discard\_data\_per\_eye\_2d(), tobi\_calibration\_clear(), tobi\_calibration\_compute\_and\_apply(), tobi\_calibration\_compute\_and\_apply\_per\_eye(), tobi\_calibration\_retrieve(), tobi\_calibration\_parse(), tobi\_calibration\_apply()

#### Example

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_calibration\_collect\_data\_per\_eye\_2d

---

#### Function

Performs data collection for the specified screen coordinate, for the left, right or both eyes.

#### Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobi_calibration_collect_data_per_eye_2d( tobi_device_t* device,
    float x, float y, tobi_enabled_eye_t requested_eyes,
    tobi_enabled_eye_t collected_eyes );
```

#### Remarks

TBD - Documentation needs to be written for this function

*device* must be a pointer to a valid tobi\_device\_t instance as created by calling tobi\_device\_create.

*x* the x-coordinate (horizontal) of the point to collect calibration data for, in normalized (0 to 1) coordinates.

*y* the y-coordinate (vertical) of the point to collect calibration data for, in normalized (0 to 1) coordinates.

*requested\_eyes* specifies wich eye to collect data for: **TOBII\_ENABLED\_EYE\_LEFT**, **TOBII\_ENABLED\_EYE\_RIGHT** or **TOBII\_ENABLED\_EYE\_BOTH**

*collected\_eyes* reports back which eye data was successfully collected for: **TOBII\_ENABLED\_EYE\_LEFT**, **TOBII\_ENABLED\_EYE\_RIGHT** or **TOBII\_ENABLED\_EYE\_BOTH**

#### Return value

If the operation is successful, tobi\_calibration\_collect\_data\_per\_eye\_2d returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, tobi\_calibration\_collect\_data\_per\_eye\_2d returns one of the following:

#### ■ **TOBII\_ERROR\_INVALID\_PARAMETER**

The *device* parameter was passed in as NULL, or *requested\_eyes* was passed in as an invalid enum value.

#### ■ **TOBII\_ERROR\_CALLBACK\_IN\_PROGRESS**

The function failed because it was called from within a callback triggered from an API call such as tobi\_device\_process\_callbacks(), tobi\_calibration\_retrieve() or tobi\_enumerate\_illumination\_modes(). Calling tobi\_calibration\_collect\_data\_per\_eye\_2d from within a callback function is not supported.

#### ■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid config level license, or has been blacklisted.

#### ■ **TOBII\_ERROR\_CONNECTION\_FAILED**

The connection to the device was lost. Call tobi\_device\_reconnect() to re-establish connection.

#### ■ **TOBII\_ERROR\_CALIBRATION\_NOT\_STARTED**

A successful call to tobi\_calibration\_start has not been made before calling this function.

#### ■ **TOBII\_ERROR\_OPERATION\_FAILED**

The tracker failed to collect a sufficient amount of data. It is recommended to performing the operation 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\_NOT\_SUPPORTED

TBD - Documentation needs to be written for this return value

#### See also

tobii\_calibration\_start(), tobi\_calibration\_stop(), tobi\_calibration\_collect\_data\_2d(), tobi\_calibration\_collect\_data\_3d(), tobi\_calibration\_discard\_data\_2d(), tobi\_calibration\_discard\_data\_3d(), tobi\_calibration\_discard\_data\_per\_eye\_2d(), tobi\_calibration\_clear(), tobi\_calibration\_compute\_and\_apply(), tobi\_calibration\_compute\_and\_apply\_per\_eye(), tobi\_calibration\_retrieve(), tobi\_calibration\_parse(), tobi\_calibration\_apply()

#### Example

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_calibration\_discard\_data\_2d

---

#### Function

Discards all data collected for the specified screen coordinate.

#### Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobi_calibration_discard_data_2d( tobi_device_t* device,
float x, float y );
```

#### Remarks

TBD - Documentation needs to be written for this function

*device* must be a pointer to a valid tobi\_device\_t instance as created by calling tobi\_device\_create.

*x* the x-coordinate (horizontal) of the point to discard data for, as specified in a prior call to tobi\_calibration\_collect\_data\_2d.

*y* the y-coordinate (vertical) of the point to discard data for, as specified in a prior call to tobi\_calibration\_collect\_data\_2d.

#### Return value

If the operation is successful, tobi\_calibration\_discard\_data\_2d returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, tobi\_calibration\_discard\_data\_2d 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 tobi\_device\_process\_callbacks(), tobi\_calibration\_retrieve() or tobi\_enumerate\_illumination\_modes(). Calling tobi\_calibration\_discard\_data\_2d from within a callback function is not supported.

#### ■ TOBII\_ERROR\_INSUFFICIENT\_LICENSE

The provided license was not a valid config level license, or has been blacklisted.

#### ■ TOBII\_ERROR\_CONNECTION\_FAILED

The connection to the device was lost. Call tobi\_device\_reconnect() to re-establish connection.

#### ■ TOBII\_ERROR\_CALIBRATION\_NOT\_STARTED

A successful call to tobi\_calibration\_start has not been made before calling this function.

#### ■ 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\_calibration\_start(), tobi\_calibration\_stop(), tobi\_calibration\_collect\_data\_2d(), tobi\_calibration\_collect\_data\_3d(), tobi\_calibration\_collect\_data\_per\_eye\_2d(), tobi\_calibration\_discard\_data\_3d(), tobi\_calibration\_discard\_data\_per\_eye\_2d(), tobi\_calibration\_clear(), tobi\_calibration\_compute\_and\_apply(), tobi\_calibration\_compute\_and\_apply\_per\_eye(), tobi\_calibration\_retrieve(), tobi\_calibration\_parse(), tobi\_calibration\_apply()

**Example** See `tobii_calibration_collect_data_2d()`.

## tobii\_calibration\_discard\_data\_3d

---

<b>Function</b>	Discards all data collected for the specified 3d coordinate.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_calibration_discard_data_3d( tobii_device_t* device, float x, float y, float z );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>x</i> the x-coordinate (horizontal) of the point to discard data for, as specified in a prior call to <code>tobii_calibration_collect_data_3d</code>.</p> <p><i>y</i> the y-coordinate (vertical) of the point to discard data for, as specified in a prior call to <code>tobii_calibration_collect_data_3d</code>.</p> <p><i>z</i> the z-coordinate (depth) of the point to discard data for, as specified in a prior call to <code>tobii_calibration_collect_data_3d</code>.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_calibration_discard_data_3d</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_calibration_discard_data_3d</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameter was passed in as NULL.</li><li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_calibration_discard_data_3d</code> from within a callback function is not supported.</li><li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> The provided license was not a valid config level license, or has been blacklisted.</li><li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</li><li>■ <b>TOBII_ERROR_CALIBRATION_NOT_STARTED</b> A successful call to <code>tobii_calibration_start</code> has not been made before calling this function.</li><li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li></ul>
<b>See also</b>	<code>tobii_calibration_start()</code> , <code>tobii_calibration_stop()</code> , <code>tobii_calibration_collect_data_2d()</code> , <code>tobii_calibration_collect_data_3d()</code> , <code>tobii_calibration_collect_data_per_eye_2d()</code> , <code>tobii_calibration_discard_data_2d()</code> , <code>tobii_calibration_discard_data_per_eye_2d()</code> , <code>tobii_calibration_clear()</code> , <code>tobii_calibration_compute_and_apply()</code> , <code>tobii_calibration_compute_and_apply_per_eye()</code> , <code>tobii_calibration_retrieve()</code> , <code>tobii_calibration_parse()</code> , <code>tobii_calibration_apply()</code>
<b>Example</b>	See <code>tobii_calibration_collect_data_3d()</code> .

## tobii\_calibration\_discard\_data\_per\_eye\_2d

---

<b>Function</b>	Discards all data collected by a corresponding call to <code>tobii_calibration_collect_data_per_eye_2d</code> .
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_calibration_discard_data_per_eye_2d( tobii_device_t* device, float x, float y, tobii_enabled_eye_t eyes );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p>

*x* the x-coordinate (horizontal) of the point to discard data for, as specified in a prior call to `tobii_calibration_collect_data_per_eye_2d`.

*y* the y-coordinate (vertical) of the point to discard data for, as specified in a prior call to `tobii_calibration_collect_data_per_eye_2d`.

*eyes* specifies which eye to discard data for: **TOBII\_ENABLED\_EYE\_LEFT**, **TOBII\_ENABLED\_EYE\_RIGHT** or **TOBII\_ENABLED\_EYE\_BOTH**, which should match the value passed in the corresponding `tobii_calibration_collect_data_per_eye_2d` call.

**Return value**

If the operation is successful, `tobii_calibration_discard_data_per_eye_2d` returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_calibration_discard_data_per_eye_2d` returns one of the following:

- **TOBII\_ERROR\_INVALID\_PARAMETER**

The *device* parameter was passed in as NULL, *eyes* was passed in as an invalid enum value.

- **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()` or `tobii_enumerate_illumination_modes()`. Calling `tobii_calibration_discard_data_per_eye_2d` from within a callback function is not supported.

- **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid config level license, or has been blacklisted.

- **TOBII\_ERROR\_CONNECTION\_FAILED**

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

- **TOBII\_ERROR\_CALIBRATION\_NOT\_STARTED**

A successful call to `tobii_calibration_start` has not been made before calling this function.

- **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\_NOT\_SUPPORTED**

TBD - Documentation needs to be written for this return value

**See also**

`tobii_calibration_start()`, `tobii_calibration_stop()`, `tobii_calibration_collect_data_2d()`, `tobii_calibration_collect_data_3d()`, `tobii_calibration_collect_data_per_eye_2d()`, `tobii_calibration_discard_data_2d()`, `tobii_calibration_discard_data_3d()`, `tobii_calibration_clear()`, `tobii_calibration_compute_and_apply()`, `tobii_calibration_compute_and_apply_per_eye()`, `tobii_calibration_retrieve()`, `tobii_calibration_parse()`, `tobii_calibration_apply()`

**Example**

See `tobii_calibration_collect_data_per_eye_2d()`.

## tobii\_calibration\_clear

---

**Function**

Resets the calibration. Also performed internally when calling `tobii_calibration_start`.

**Syntax**

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

**Remarks**

TBD - Documentation needs to be written for this function

*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_calibration_clear` returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_calibration_clear` 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()` or `tobii_enumerate_illumination_modes()`.

Calling `tobii_calibration_clear` from within a callback function is not supported.

■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid config level license, or has been blacklisted.

■ **TOBII\_ERROR\_CONNECTION\_FAILED**

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

■ **TOBII\_ERROR\_CALIBRATION\_NOT\_STARTED**

A successful call to `tobii_calibration_start` has not been made before calling this function.

■ **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_calibration_start()`, `tobii_calibration_stop()`, `tobii_calibration_collect_data_2d()`, `tobii_calibration_collect_data_3d()`, `tobii_calibration_collect_data_per_eye_2d()`, `tobii_calibration_discard_data_2d()`, `tobii_calibration_discard_data_3d()`, `tobii_calibration_discard_data_per_eye_2d()`, `tobii_calibration_compute_and_apply()`, `tobii_calibration_compute_and_apply_per_eye()`, `tobii_calibration_retrieve()`, `tobii_calibration_parse()`, `tobii_calibration_apply()`

**Example**

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_calibration\_compute\_and\_apply

---

**Function** Computes a calibration based on data collected so far, and applies the resulting calibration to the device.

**Syntax**

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

**Remarks** TBD - Documentation needs to be written for this function

*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_calibration_compute_and_apply` returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_calibration_compute_and_apply` 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()` or `tobii_enumerate_illumination_modes()`. Calling `tobii_calibration_compute_and_apply` from within a callback function is not supported.

■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid config level license, or has been blacklisted.

■ **TOBII\_ERROR\_CONNECTION\_FAILED**

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

■ **TOBII\_ERROR\_CALIBRATION\_NOT\_STARTED**

A successful call to `tobii_calibration_start` has not been made before calling this function.

■ **TOBII\_ERROR\_OPERATION\_FAILED**

Not enough data collected to compute calibration.

■ **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\_calibration\_start(), tobii\_calibration\_stop(), tobii\_calibration\_collect\_data\_2d(),  
tobii\_calibration\_collect\_data\_3d(), tobii\_calibration\_collect\_data\_per\_eye\_2d(),  
tobii\_calibration\_discard\_data\_2d(), tobii\_calibration\_discard\_data\_3d(),  
tobii\_calibration\_discard\_data\_per\_eye\_2d(), tobii\_calibration\_clear(),  
tobii\_calibration\_compute\_and\_apply\_per\_eye(), tobii\_calibration\_retrieve(), tobii\_calibration\_parse(),  
tobii\_calibration\_apply()

**Example**      See tobii\_calibration\_collect\_data\_2d().

## tobii\_calibration\_compute\_and\_apply\_per\_eye

---

**Function**      Computes a calibration based on data collected so far, using tobii\_calibration\_collect\_data\_per\_eye\_2d, and applies the resulting calibration to the device.

**Syntax**

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_calibration_compute_and_apply_per_eye( tobii_device_t* device,
    tobii_enabled_eye_t* calibrated_eyes );
```

**Remarks**      TBD - Documentation needs to be written for this function

*device* must be a pointer to a valid tobii\_device\_t instance as created by calling tobii\_device\_create.

*calibrated\_eyes* receives information about which eyes were successfully calibrated:

**TOBII\_ENABLED\_EYE\_LEFT, TOBII\_ENABLED\_EYE\_RIGHT or TOBII\_ENABLED\_EYE\_BOTH**

**Return value**      If the operation is successful, tobii\_calibration\_compute\_and\_apply\_per\_eye returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, tobii\_calibration\_compute\_and\_apply\_per\_eye 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() or tobii\_enumerate\_illumination\_modes(). Calling tobii\_calibration\_compute\_and\_apply\_per\_eye from within a callback function is not supported.

■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid config level license, or has been blacklisted.

■ **TOBII\_ERROR\_CONNECTION\_FAILED**

The connection to the device was lost. Call tobii\_device\_reconnect() to re-establish connection.

■ **TOBII\_ERROR\_CALIBRATION\_NOT\_STARTED**

A successful call to tobii\_calibration\_start has not been made before calling this function.

■ **TOBII\_ERROR\_OPERATION\_FAILED**

Not enough data collected to compute calibration.

■ **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\_calibration\_start(), tobii\_calibration\_stop(), tobii\_calibration\_collect\_data\_2d(),  
tobii\_calibration\_collect\_data\_3d(), tobii\_calibration\_collect\_data\_per\_eye\_2d(),  
tobii\_calibration\_discard\_data\_2d(), tobii\_calibration\_discard\_data\_3d(),  
tobii\_calibration\_discard\_data\_per\_eye\_2d(), tobii\_calibration\_clear(),  
tobii\_calibration\_compute\_and\_apply(), tobii\_calibration\_retrieve(), tobii\_calibration\_parse(),  
tobii\_calibration\_apply()

**Example**      See tobii\_calibration\_collect\_data\_per\_eye\_2d().

## tobii\_calibration\_retrieve

---

<b>Function</b>	Retrieves the currently applied calibration from the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_calibration_retrieve( tobii_device_t* device,     tobii_data_receiver_t receiver, void* user_data );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>receiver</i> is a function pointer to a function with the prototype:</p> <pre>void data_receiver( void const* data, size_t size, void* user_data )</pre> <p>This function will be called with the retrieved calibration data. It is called with the following parameters:</p> <ul style="list-style-type: none"><li>▪ <i>data</i> The calibration data read from device. This pointer will be invalid after returning from the function, so ensure you make a copy of the data rather than storing the pointer directly.</li><li>▪ <i>size</i> The size of the calibration data, in bytes.</li><li>▪ <i>user_data</i> This is the custom pointer passed to <code>tobii_calibration_retrieve</code>.</li></ul> <p><i>user_data</i> custom pointer which will be passed unmodified to the receiver function.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_calibration_retrieve</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_calibration_retrieve</code> returns one of the following:</p> <ul style="list-style-type: none"><li>▪ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> or <i>receiver</i> parameter was passed in as NULL.</li><li>▪ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_calibration_retrieve</code> from within a callback function is not supported.</li><li>▪ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> The provided license was not a valid config level license, or has been blacklisted.</li><li>▪ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li><li>▪ TBD - Other possible error values currently unknown</li></ul>
<b>See also</b>	<code>tobii_calibration_start()</code> , <code>tobii_calibration_stop()</code> , <code>tobii_calibration_collect_data_2d()</code> , <code>tobii_calibration_collect_data_3d()</code> , <code>tobii_calibration_collect_data_per_eye_2d()</code> , <code>tobii_calibration_discard_data_2d()</code> , <code>tobii_calibration_discard_data_3d()</code> , <code>tobii_calibration_discard_data_per_eye_2d()</code> , <code>tobii_calibration_clear()</code> , <code>tobii_calibration_compute_and_apply()</code> , <code>tobii_calibration_compute_and_apply_per_eye()</code> , <code>tobii_calibration_parse()</code> , <code>tobii_calibration_apply()</code>
<b>Example</b>	<pre>#include &lt;tobii/tobii_config.h&gt;  int main() {     // TODO: Implement example }</pre>

## tobii\_calibration\_parse

---

<b>Function</b>	Extracts data about calibration points from the specified calibration.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_calibration_parse( tobii_api_t* api, void const* data,     size_t data_size, tobii_calibration_point_data_receiver_t receiver,     void* user_data );</pre>

<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>api</i> must be a pointer to a valid <code>tobii_api_t</code> instance as created by calling <code>tobii_api_create</code>.</p> <p><i>data</i> is the calibration data retrieved by <code>tobii_calibration_retrieve()</code>.</p> <p><i>data_size</i> is the size of the data retrieved by <code>tobii_calibration_retrieve()</code>.</p> <p><i>receiver</i> is a function pointer to a function with the prototype:</p> <pre>void receiver( tobii_calibration_point_data_t const* point_data, void* user_data )</pre> <p>This function will be called for each parsed point from the calibration. It is called with the following parameters:</p> <ul style="list-style-type: none"> <li>▪ <i>point_data</i> A pointer to a struct containing all the data related to a calibration point. TBD - document the meaning of each field</li> <li>▪ <i>user_data</i> This is the custom pointer sent in to <code>tobii_calibration_parse</code>.</li> </ul> <p><i>user_data</i> custom pointer which will be passed unmodified to the receiver function.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_calibration_parse</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_calibration_parse</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>▪ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>api</i>, <i>data</i> or <i>receiver</i> parameters were passed in as NULL, or <i>data_size</i> parameter was less than 8.</li> <li>▪ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_calibration_parse</code> from within a callback function is not supported.</li> <li>▪ <b>TOBII_OPERATION_FAILED</b> The data being parsed was not a valid calibration.</li> </ul>
<b>See also</b>	<p><code>tobii_calibration_start()</code>, <code>tobii_calibration_stop()</code>, <code>tobii_calibration_collect_data_2d()</code>, <code>tobii_calibration_collect_data_3d()</code>, <code>tobii_calibration_collect_data_per_eye_2d()</code>, <code>tobii_calibration_discard_data_2d()</code>, <code>tobii_calibration_discard_data_3d()</code>, <code>tobii_calibration_discard_data_per_eye_2d()</code>, <code>tobii_calibration_clear()</code>, <code>tobii_calibration_compute_and_apply()</code>, <code>tobii_calibration_compute_and_apply_per_eye()</code>, <code>tobii_calibration_retrieve()</code>, <code>tobii_calibration_apply()</code></p>
<b>Example</b>	<pre>#include &lt;tobii/tobii_config.h&gt;  int main() {     // TODO: Implement example }</pre>

## tobii\_calibration\_apply

<b>Function</b>	Applies the specified calibration to the device, making it the current calibration.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_calibration_apply( tobii_device_t* device,     void const* data, size_t size );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>data</i> is the calibration data which has previously been retrieved by calling <code>tobii_calibration_retrieve()</code></p> <p><i>size</i> is the size of the calibration data which has previously been retrieved by calling <code>tobii_calibration_retrieve()</code></p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_calibration_apply</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_calibration_apply</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>▪ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> or <i>data</i> parameters were passed in as NULL, or the <i>data_size</i> parameter was not greater than 0.</li> </ul>



#### ■ **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()` or `tobii_enumerate_illumination_modes()`. Calling `tobii_calibration_apply` from within a callback function is not supported.

#### ■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid config level license, or has been blacklisted.

#### ■ **TOBII\_ERROR\_CONNECTION\_FAILED**

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

#### ■ **TOBII\_ERROR\_CALIBRATION\_BUSY**

The device is currently being calibrated. `tobii_calibration_apply` can not be called while calibrating the device.

#### ■ **TOBII\_ERROR\_OPERATION\_FAILED**

The provided calibration could not be applied to the device.

#### ■ **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_calibration_start()`, `tobii_calibration_stop()`, `tobii_calibration_collect_data_2d()`, `tobii_calibration_collect_data_3d()`, `tobii_calibration_collect_data_per_eye_2d()`, `tobii_calibration_discard_data_2d()`, `tobii_calibration_discard_data_3d()`, `tobii_calibration_discard_data_per_eye_2d()`, `tobii_calibration_clear()`, `tobii_calibration_compute_and_apply()`, `tobii_calibration_compute_and_apply_per_eye()`, `tobii_calibration_retrieve()`, `tobii_calibration_parse()`

#### Example

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_get\_geometry\_mounting

---

**Function** Retrieves the geometry mounting of the device.

**Syntax**

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_get_geometry_mounting( tobii_device_t* device,
    tobii_geometry_mounting_t* geometry_mounting );
```

**Remarks** TBD - Documentation needs to be written for this function

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

*geometry\_mounting* must be a valid pointer to a `tobii_geometry_mounting_t` instance which will receive the result.

**Return value** If the operation is successful, `tobii_get_geometry_mounting` returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_get_geometry_mounting` returns one of the following:

#### ■ **TOBII\_ERROR\_INVALID\_PARAMETER**

The *device* or *geometry\_mounting* parameters were 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()` or `tobii_enumerate_illumination_modes()`. Calling `tobii_get_geometry_mounting` from within a callback function is not supported.

#### ■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid license, or has been blacklisted.

#### ■ **TOBII\_ERROR\_INTERNAL**

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

- TBD - Other possible error values currently unknown

**See also**      `tobii_calculate_display_area_basic()`

**Example**

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_get\_display\_area

---

<b>Function</b>	Retrieves the current display area from the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_get_display_area( tobii_device_t* device,     tobii_display_area_t* display_area );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>display_area</i> must be a valid pointer to a <code>tobii_display_area_t</code> instance which will receive the result.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_get_display_area</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_get_display_area</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> or <i>display_area</i> parameters were passed in as NULL.</li><li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_get_display_area</code> from within a callback function is not supported.</li><li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> The provided license was not a valid license, or has been blacklisted.</li><li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li><li>■ TBD - Other possible error values currently unknown</li></ul>
<b>See also</b>	<code>tobii_set_display_area()</code> , <code>tobii_calculate_display_area_basic()</code>
<b>Example</b>	<pre>#include &lt;tobii/tobii_config.h&gt;  int main() {     // TODO: Implement example }</pre>

## tobii\_set\_display\_area

---

<b>Function</b>	Applies the specified display area setting to the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_set_display_area( tobii_device_t* device,     tobii_display_area_t const* display_area );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>display_area</i> must be a valid pointer to a <code>tobii_display_area_t</code> which is correctly initialize, for example by callin</p>

tobii\_calculate\_display\_area\_basic().

<b>Return value</b>	<p>If the operation is successful, <code>tobii_set_display_area</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_set_display_area</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> or <i>display_area</i> parameters were passed in as NULL.</li><li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_set_display_area</code> from within a callback function is not supported.</li><li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> The provided license was not a valid config level license, or has been blacklisted.</li><li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li><li>■ TBD - Other possible error values currently unknown</li></ul>
<b>See also</b>	<code>tobii_get_display_area()</code> , <code>tobii_calculate_display_area_basic()</code>
<b>Example</b>	<pre>#include &lt;tobii/tobii_config.h&gt;  int main() {     // TODO: Implement example }</pre>

## tobii\_calculate\_display\_area\_basic

---

<b>Function</b>	Calculates a basic display area configuration based on screen size and geometry mounting.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_calculate_display_area_basic( tobii_api_t* api,     float width_mm, float height_mm, float offset_x_mm,     tobii_geometry_mounting_t const* geometry_mounting,     tobii_display_area_t* display_area );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>width_mm</i> is the width of the display device in millimeters.</p> <p><i>height_mm</i> is the height of the display device in millimeters.</p> <p><i>offset_x</i> is the offset of the eye tracker from the center of the display device in the x-axis, in millimeters.</p> <p><i>geometry_mounting</i> is the geometry mounting information as received by <code>tobii_get_geometry_mounting()</code></p> <p><i>display_area</i> must be a valid pointer to a <code>tobii_display_area_t</code> instance which will receive the result.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_calculate_display_area_basic</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_calculate_display_area_basic</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>api</i>, <i>geometry_mounting</i> or <i>display_area</i> parameters was passed in as NULL.</li></ul>
<b>See also</b>	<code>tobii_get_display_area()</code> , <code>tobii_get_geometry_mounting()</code> ,
<b>Example</b>	<pre>#include &lt;tobii/tobii_config.h&gt;  int main() {     // TODO: Implement example }</pre>

## tobii\_get\_device\_name

---

<b>Function</b>	Retrieves the users nickname for the device, if it has been set.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_get_device_name( tobii_device_t* device,     tobii_device_name_t* device_name );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>device_name</i> must be a valid pointer to a <code>tobii_device_name_t</code> instance which will receive the result.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_get_device_name</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_get_device_name</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> or <i>device_name</i> parameters were passed in as NULL.</li><li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_get_device_name</code> from within a callback function is not supported.</li><li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> The provided license was not a valid license, or has been blacklisted.</li><li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li><li>■ TBD - Other possible error values currently unknown</li></ul>
<b>See also</b>	<code>tobii_set_device_name()</code>
<b>Example</b>	<pre>#include &lt;tobii/tobii_config.h&gt;  int main() {     // TODO: Implement example }</pre>

## tobii\_set\_device\_name

---

<b>Function</b>	Sets a user nickname for the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_set_device_name( tobii_device_t* device,     tobii_device_name_t const device_name );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>device_name</i> a <code>tobii_device_name_t</code> containing the name to be applied.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_set_device_name</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_set_device_name</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> or <i>device_name</i> parameters were passed in as NULL.</li><li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_set_device_name</code> from within a callback function is not supported.</li></ul>

- **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid config level license, or has been blacklisted.

- **TOBII\_ERROR\_INTERNAL**

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

- TBD - Other possible error values currently unknown

**See also**      `tobii_get_device_name()`

**Example**

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_enumerate\_output\_frequencies

---

**Function**      Lists all valid output frequencies for the device.

**Syntax**

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_enumerate_output_frequencies( tobii_device_t* device,
    tobii_output_frequency_receiver_t receiver, void* user_data );
```

**Remarks**      TBD - Documentation needs to be written for this function

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

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

```
void receiver( ( float output_frequency, void* user_data ) )
```

This function will be called for each available output frequency. It is called with the following parameters:

- *output\_frequency* The frequency in Hz.
- *user\_data* This is the custom pointer sent in to `tobii_enumerate_output_frequencies`.

*user\_data* custom pointer which will be passed unmodified to the receiver function.

**Return value**      If the operation is successful, `tobii_enumerate_output_frequencies` returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_enumerate_output_frequencies` returns one of the following:

- **TOBII\_ERROR\_INVALID\_PARAMETER**

The *device* or *receiver* parameters were 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()` or `tobii_enumerate_illumination_modes()`. Calling `tobii_get_geometry_mounting` from within a callback function is not supported.

- **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid license, or has been blacklisted.

- **TOBII\_ERROR\_INTERNAL**

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

- TBD - Other possible error values currently unknown

**See also**      `tobii_set_output_frequency()`, `tobii_get_output_frequency()`

**Example**

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_set\_output\_frequency

---

<b>Function</b>	Configures the device to run at the specified output frequency.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobi_i_set_output_frequency( tobii_device_t* device, float output_frequency );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>output_frequency</i> the frequency to apply.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_set_output_frequency</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_set_output_frequency</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> parameters were passed in as NULL, or <i>output_frequency</i> is lower than 0.</li><li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_set_output_frequency</code> from within a callback function is not supported.</li><li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> The provided license was not a valid config level license, or has been blacklisted.</li><li>■ <b>TOBII_ERROR_OPERATION_FAILED</b> The function failed because it was called while the device was in calibration mode.</li><li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li><li>■ TBD - Other possible error values currently unknown</li></ul>
<b>See also</b>	<code>tobii_get_output_frequency()</code> , <code>tobii_enumerate_output_frequencies()</code>
<b>Example</b>	<pre>#include &lt;tobii/tobii_config.h&gt;  int main() {     // TODO: Implement example }</pre>

## tobii\_get\_output\_frequency

---

<b>Function</b>	Queries the current output frequency of the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobi_i_get_output_frequency( tobii_device_t* device, float* output_frequency );</pre>
<b>Remarks</b>	<p>TBD - Documentation needs to be written for this function</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>output_frequency</i> is a valid pointer to a float which will receive the current output frequency.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_get_output_frequency</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_get_output_frequency</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> or <i>output_frequency</i> parameters were passed in as NULL.</li><li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as</li></ul>

tobii\_device\_process\_callbacks(), tobii\_calibration\_retrieve() or tobii\_enumerate\_illumination\_modes(). Calling tobii\_get\_output\_frequency from within a callback function is not supported.

■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid license, or has been blacklisted.

■ **TOBII\_ERROR\_INTERNAL**

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

■ TBD - Other possible error values currently unknown

**See also** tobii\_set\_output\_frequency(), tobii\_enumerate\_output\_frequencies()

**Example**

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_get\_display\_id

---

**Function** Retrieves the current display id from the device.

**Syntax**

```
#include <tobii/tobii_config.h>
tobii_error t tobii_get_display_id( tobii_device_t* device,
    tobii_display_id_t* display_id );
```

**Remarks** The format of the display id is application specific and transparent to stream engine. Stream engine does not impose that any specific format be used. Hence, encoding and decoding the display id is up to the producer and the consumer to decide.

*device* must be a pointer to a valid tobii\_device\_t instance as created by calling tobii\_device\_create.

*display\_id* must be a valid pointer to a tobii\_display\_id\_t instance which will receive the result. The maximum size of the returned tobii\_display\_id is 256 characters and it is '\0'-terminated. The default value is all '\0' characters, meaning that the display id has not yet been set.

**Return value** If the operation is successful, tobii\_get\_display\_id returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, tobii\_get\_display\_id returns one of the following:

■ **TOBII\_ERROR\_INVALID\_PARAMETER**

The *device* or *display\_id* parameters were 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() or tobii\_enumerate\_illumination\_modes(). Calling tobii\_get\_display\_id from within a callback function is not supported.

■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid license, or has been blacklisted.

■ **TOBII\_ERROR\_INTERNAL**

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

■ TBD - Other possible error values currently unknown

**See also** tobii\_set\_display\_id()

**Example**

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

int main()
{
    // TODO: Implement example
}
```

## tobii\_set\_display\_id

---

<b>Function</b>	Applies the specified display area setting to the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_set_display_id( tobii_device_t* device,     tobii_display_id_t const* display_id );</pre>
<b>Remarks</b>	<p>The format of the display id is application specific and transparent to stream engine. Stream engine does not impose that any specific format be used. Hence, encoding and decoding the display id is up to the producer and the consumer to decide.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>display_id</i> a valid <code>tobii_display_id_t</code> containing the display id to be applied. The maximum size of the device id is 256 characters and it shall be '\0'-terminated.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_set_display_id</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_set_display_id</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> or <i>display_id</i> parameters were passed in as NULL.</li><li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_set_display_id</code> from within a callback function is not supported.</li><li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> The provided license was not a valid config level license, or has been blacklisted.</li><li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li><li>■ TBD - Other possible error values currently unknown</li></ul>
<b>See also</b>	<code>tobii_get_display_id()</code>
<b>Example</b>	<pre>#include &lt;tobii/tobii_config.h&gt;  int main() {     // TODO: Implement example }</pre>

## tobii\_display\_id\_subscribe

---

<b>Function</b>	Start listening for changes to the display ID property.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_display_id_subscribe( tobii_device_t* device,     tobii_display_id_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	<p>This subscription is for receiving notifications on changes to the display ID property.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>callback</i> is a function pointer to a function with the prototype:</p> <pre>void tobii_display_id_callback( tobii_display_id_t const* display_id, void* user_data )</pre> <p>This function will be called when the display id property has changed. It is called with the following parameters:</p> <ul style="list-style-type: none"><li>■ <i>display_id</i> This is a pointer to a char array of size 256 containing the '\0'-terminated device id.</li></ul>



- *user\_data* This is a pointer to custom user data sent in when registering the callback in the call to `tobii_display_id_subscribe()`.

*user\_data* pointer to custom user data, which will be passed unmodified to the callback function.

## Return value

If the operation is successful, `tobii_display_id_subscribe` returns **TOBII\_ERROR\_NO\_ERROR**. If the call fails, `tobii_display_id_subscribe` returns one of the following error codes:

### ▪ TOBII\_ERROR\_INVALID\_PARAMETER

One or more of the *device* and *callback* parameters were passed in as NULL. *device* must be a valid `tobii_device_t` pointer as created by `tobii_device_create`, and *callback* must be a valid pointer to a `tobii_display_id_callback_t` function.

### ▪ TOBII\_ERROR\_INSUFFICIENT\_LICENSE

The provided license was not valid, or has been blacklisted.

### ▪ TOBII\_ERROR\_NOT\_SUPPORTED

The functionality is not supported on this device.

### ▪ TOBII\_ERROR\_ALREADY\_SUBSCRIBED

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

### ▪ TOBII\_ERROR\_CALLBACK\_IN\_PROGRESS

This function is called from a tobii call back function which is not allowed.

- **TOBII\_ERROR\_OPERATION\_FAILED** The display id property change notification subscribe operation 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_display_id_unsubscribe()`, `tobii_wait_for_callbacks()`, `tobii_device_process_callbacks()`

## Example

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

void display_id_callback( tobii_display_id_t const display_id, void* user_data )
{
    printf( "Display id: %s\n", display_id );
}

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, TOBII_FIELD_OF_USE_INTERACTIVE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_display_id_subscribe( device, display_id_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( 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_display_id_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\_display\_id\_unsubscribe

---

<b>Function</b>	Stops listening to notifications on changes to display id property that was subscribed to by a call to <code>tobii_display_id_subscribe()</code>
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_config.h&gt; tobii_error_t tobii_display_id_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	<i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code> .
<b>Return value</b>	If the operation is successful, <code>tobii_display_id_unsubscribe</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_display_id_unsubscribe</code> returns an error code specific to the device.
<b>See also</b>	<code>tobii_display_id_subscribe()</code>
<b>Example</b>	See <code>tobii_display_id_subscribe()</code>

# tobii\_advanced.h

The tobii\_advanced.h file contains advanced features that require a professional license to use.

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\_data\_subscribe

---

<b>Function</b>	Starts the gaze data stream.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_advanced.h&gt; tobii_error_t tobii_gaze_data_subscribe( tobii_device_t* device,     tobii_gaze_data_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	To be able to call this function, the <i>device</i> should have been created with a minimum license level of Professional feature group.

*device* must be a pointer to a valid tobii\_device\_t as created by calling tobii\_device\_create.

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

```
void gaze_data_callback( tobii_gaze_data_t const* gaze_data, void* user_data )
```

Older devices using the deprecated 0-4 scale to determine validity will have the value map to the new binary scale accordingly:

```
0 - TOBII_VALIDITY_VALID
1 - TOBII_VALIDITY_VALID
2 - TOBII_VALIDITY_INVALID
3 - TOBII_VALIDITY_INVALID
4 - TOBII_VALIDITY_INVALID
```

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

- *gaze\_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 gaze data was captured in microseconds (us). It is generated on the device responsible for capturing the data. *timestamp\_system\_us* is generated using this value. The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values.
  - *timestamp\_system\_us* Timestamp value for when the gaze data 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.
  - *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\_from\_eye\_tracker\_mm* An array of three floats, for the x, y and z coordinate of the gaze origin point of the eye of the user, as measured in millimeters from the center of the device.
    - *eye\_position\_validity* **TOBII\_VALIDITY\_INVALID** if *eye\_position\_in\_track\_box\_normalized\_xyz* is not valid for this frame, **TOBII\_VALIDITY\_VALID** if it is.
    - *eye\_position\_in\_track\_box\_normalized\_xyz* An array of three floats, for the x, y and z coordinate of the gaze origin point of the eye of the user, as measured in the normalized distance of the device track box.
    - *gaze\_point\_validity* **TOBII\_VALIDITY\_INVALID** if *gaze\_point\_from\_eye\_tracker\_mm* and *gaze\_point\_on\_display\_normalized* are not valid for this frame, **TOBII\_VALIDITY\_VALID** if they are.

- *gaze\_point\_from\_eye\_tracker\_mm* An array of three floats, for the x, y and z coordinate of the gaze point that the user is currently looking, as measured in millimeters from the center of the device.
- *gaze\_point\_on\_display\_normalized* The horizontal and vertical 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.
- *eyeball\_center\_validity* **TOBII\_VALIDITY\_INVALID** if *eyeball\_center\_from\_eye\_tracker\_mm* is not valid for this frame, **TOBII\_VALIDITY\_VALID** if it is.
- *eyeball\_center\_from\_eye\_tracker\_mm* An array of three floats, for the x, y and z coordinate of the center of the eyeball, as measured in millimeters from the center of the device.
- *pupil\_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.
- *right* This is another instance of the same struct as in *left*, but which holds data related to the right eye of 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 callback.

**Return value** If the call was successful **TOBII\_ERROR\_NO\_ERROR** will be returned. If the call fails, *tobii\_gaze\_data\_subscribe* returns an error code specific to the device.

**See Also** *tobii\_gaze\_data\_unsubscribe()*

## tobii\_gaze\_data\_unsubscribe

<b>Function</b>	Stops the gaze data stream.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_advanced.h&gt; tobii_gaze_data_unsubscribe( tobii_device_t* device );</pre>
<b>Remarks</b>	To be able to call this function, the <i>device</i> should have been created with a minimum license level of Professional feature group. <i>device</i> must be a pointer to a valid <i>tobii_device_t</i> as created by calling <i>tobii_device_create</i> .
<b>Return value</b>	If the call was successful <b>TOBII_ERROR_NO_ERROR</b> will be returned. If the call fails, <i>tobii_gaze_data_unsubscribe</i> returns an error code specific to the device.
<b>See Also</b>	<i>tobii_gaze_data_subscribe()</i>
<b>Example</b>	<pre>#include "tobii/tobii.h" #include "tobii/tobii_licensing.h" #include "tobii/tobii_advanced.h"  #include &lt;stdio.h&gt; #include &lt;assert.h&gt;  static void tobii_gaze_data_callback( tobii_gaze_data_t const* gaze_data, void* user_data ) {     (void)user_data;     if( gaze_data-&gt;right.gaze_point_validity == TOBII_VALIDITY_VALID )         printf( "Gaze point (right): %f, %f\n",             gaze_data-&gt;right.gaze_point_on_display_normalized_xy[ 0 ],             gaze_data-&gt;right.gaze_point_on_display_normalized_xy[ 1 ] );     else         printf( "Gaze point (right): INVALID\n");      if( gaze_data-&gt;left.gaze_point_validity == TOBII_VALIDITY_VALID )         printf( "Gaze point (left): %f, %f\n",             gaze_data-&gt;left.gaze_point_on_display_normalized_xy[ 0 ],             gaze_data-&gt;left.gaze_point_on_display_normalized_xy[ 1 ] );     else         printf( "Gaze point (left): INVALID\n"); }  int main() {     tobii_api_t* api;     tobii_error_t error = tobii_api_create( &amp;api, NULL, NULL );</pre>

```

assert( error == TOBII_ERROR_NO_ERROR );

tobii_device_t* device;
char url[ 256 ] = { 0 };
printf( "Enter url to the eye tracker (don't forget prefix tobii-ttp:// or tet-tcp://):\n" );
scanf( "%255s", url );
error = tobii_device_create( api, url, TOBII_FIELD_OF_USE_INTERACTIVE, &device );      // if not using
a pro tracker use tobii_device_create_ex with Professional license
assert( error == TOBII_ERROR_NO_ERROR );

error = tobii_gaze_data_subscribe( device, tobii_gaze_data_callback, 0 );
assert( error == TOBII_ERROR_NO_ERROR );

int is_running = 10; // in this sample, exit after some iterations
while( --is_running > 0 )
{
    error = tobii_wait_for_callbacks( 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_data_unsubscribe( device );
assert( error == TOBII_ERROR_NO_ERROR );

tobii_device_destroy( device );
tobii_api_destroy( api );

return 0;
}

```

## tobii\_digital\_syncport\_subscribe

---

<b>Function</b>	The digital syncport data stream subscription provides a sparse stream of the device's external port data in sync with the device clock. This stream will provide new data when the syncport data value changes. Each change on the port is timestamped with the same clock as the gaze data.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_advanced.h&gt; tobii_error_t tobii_digital_syncport_subscribe( tobii_device_t* device,     tobii_digital_syncport_callback_t callback, void* user_data );</pre>
<b>Remarks</b>	<p>To be able to call this function, the <i>device</i> should have been created with a minimum license level of Professional feature group.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> as created by calling <code>tobii_device_create</code>.</p> <p><i>callback</i> is a function pointer to a function with the prototype:</p> <pre>void digital_syncport_callback( uint32_t signal, int64_t timestamp_tracker_us,     int64_t timestamp_system_us, void* user_data )</pre> <p>This function will be called when the syncport data value changes. It is called with the following parameters:</p> <ul style="list-style-type: none"> <li>- <i>*signal*</i> An unsigned integer from the external port. In the Spectrum device, only 8 bits are valid. Please check the hardware documentation of the relevant device for its valid bits.</li> <li>- <i>*timestamp_tracker_us*</i> Timestamp value for when the digital syncport data was captured in microseconds (us). It is generated on the device responsible for capturing the data. <i>*timestamp_system_us*</i> is generated using this value. The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values.</li> <li>- <i>*timestamp_system_us*</i> Timestamp value for when the digital syncport data 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 <code>tobii_system_clock</code> can be used to retrieve a timestamp using the same clock and same relative values as this timestamp.</li> <li>- <i>*user_data*</i> the custom pointer sent in when registering the callback.</li> </ul> <p><i>user_data</i> custom pointer which will be passed unmodified to the callback.</p>
<b>Return value</b>	If the call was successful <b>TOBII_ERROR_NO_ERROR</b> will be returned. If the call has failed one of the following error will be returned:

■ **TOBII\_ERROR\_INVALID\_PARAMETER**

The *device* parameter has been passed in as NULL.

■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not valid, or has been blacklisted.

■ **TOBII\_ERROR\_ALREADY\_SUBSCRIBED**

A subscription for digital syncport data was already made. There can only be one callback registered at a time. To change to another callback, first call `tobii_digital_syncport_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_digital_syncport_subscribe` from within a callback function is not supported.

**See Also**      `tobii_digital_syncport_unsubscribe()`

## tobii\_digital\_syncport\_unsubscribe

---

**Function**      Stops the digital syncport data stream.

**Syntax**

```
#include <tobii/tobii_advanced.h>
tobii_digital_syncport_unsubscribe( tobii_device_t* device );
```

**Remarks**      To be able to call this function, the *device* should have been created with a minimum license level of Professional feature group. *device* must be a pointer to a valid `tobii_device_t` as created by calling `tobii_device_create`.

**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* parameter has been passed in as NULL.

■ **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not valid, or has been blacklisted.

■ **TOBII\_ERROR\_NOT\_SUBSCRIBED**

A subscription for digital syncport data was not made before the call to unsubscribe.

■ **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_digital_syncport_unsubscribe` from within a callback function is not supported.

**See Also**      `tobii_digital_syncport_subscribe()`

**Example**

```
#include "tobii/tobii.h"
#include "tobii/tobii_licensing.h"
#include "tobii/tobii_advanced.h"
```

```

#include <stdio.h>
#include <assert.h>

static void tobii_digital_syncport_callback( uint32_t signal, int64_t timestamp_tracker_us,
int64_t timestamp_system_us, void* user_data )
{
    (void)timestamp_tracker_us;(void)timestamp_system_us;(void)user_data;
    printf( "Digital syncport data is %d .\n", signal & 0xff ); // only 8 bits are valid for spectrum
    tracker
}

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

    tobii_device_t* device;
    char url[ 256 ] = { 0 };
    printf( "Enter url to the eye tracker (don't forget prefix tobii-ttp:// or tet-tcp://):\n" );
    scanf( "%255s", url );
    error = tobii_device_create( api, url, TOBII_FIELD_OF_USE_INTERACTIVE, &device ); // if not using
a pro tracker use tobii_device_create_ex with Professional license
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_digital_syncport_subscribe( device, tobii_digital_syncport_callback, 0 );
    assert( error == TOBII_ERROR_NO_ERROR );

    int is_running = 10; // in this sample, exit after some iterations
    while( --is_running > 0 )
    {
        error = tobii_wait_for_callbacks( 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_digital_syncport_unsubscribe( device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}

```

## tobii\_enumerate\_face\_types

<b>Function</b>	Retreives all supported face types from a specific eye tracker.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_advanced.h&gt; tobii_error_t tobii_enumerate_face_types( tobii_device_t* device, tobii_face_type_receiver_t receiver, void* user_data );</pre>
<b>Remarks</b>	<p>A face type is here understood as a class of appearances of the face itself, such as a group of species (e.g. human or crocodile), facial features (e.g. moustache or makeup), or worn objects (e.g. glasses or hats). It is only used for situations where auto detecting such differences is difficult or dangerous.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>receiver</i> is a function pointer to a function with the prototype:</p> <pre>void face_type_receiver( const tobii_face_type_t face_type, void* user_data );</pre> <p>This function will be called for each face type found during enumeration. It is called with the following parameters:</p> <ul style="list-style-type: none"> <li>▪ <i>face_type</i> A zero terminated string representation of a face type, max 63 characters long. 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.</li> <li>▪ <i>user_data</i> This is the custom pointer sent in to <code>tobii_enumerate_face_types</code>.</li> </ul> <p><i>user_data</i> custom pointer which will be passed unmodified to the receiver function.</p>
<b>Return value</b>	If the operation is successful, <code>tobii_enumerate_face_types</code> returns <b>TOBII_ERROR_NO_ERROR</b> . If the call fails, <code>tobii_enumerate_face_types</code> returns one of the following:

- **TOBII\_ERROR\_INVALID\_PARAMETER**

The *device* or *receiver* 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()` or `tobii_enumerate_illumination_modes()`. Calling `tobii_enumerate_face_types` from within a callback function is not supported.

- **TOBII\_ERROR\_INSUFFICIENT\_LICENSE**

The provided license was not a valid consumer level license, or has been blacklisted.

- **TOBII\_ERROR\_CONNECTION\_FAILED**

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

- **TOBII\_ERROR\_NOT\_SUPPORTED**

The eye tracker does not support enumeration of face types.

- **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_get_face_type()` and `tobii_set_face_type()`

## tobii\_set\_face\_type

---

<b>Function</b>	Applies the specified face type setting to the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_advanced.h&gt; tobii_error_t tobii_set_face_type( tobii_device_t* device, tobii_face_type_t const face_type );</pre>
<b>Remarks</b>	<p>Applying a new face type causes the current personal calibration to be discarded and the tracker will revert to the built-in default calibration for the given face type. A <code>TOBII_NOTIFICATION_TYPE_FACE_TYPE_CHANGED</code> will be broadcasted to all clients notifying them that the face type has changed and that a new calibration has to be set.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>face_type</i> is a zero-terminated string representation of a specific face type setting, with a maximum length of 63 characters. Supported string values can be queried by calling the <code>tobii_enumerate_face_types()</code> function.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_set_face_type</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_set_face_type</code> returns one of the following:</p> <ul style="list-style-type: none"> <li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> <p>The <i>device</i> or <i>receiver</i> parameter was passed in as NULL.</p> </li> <li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> <p>The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_set_face_type</code> from within a callback function is not supported.</p> </li> <li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> <p>The provided license was not a valid config level license, or has been blacklisted.</p> </li> <li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> <p>The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</p> </li> <li>■ <b>TOBII_ERROR_OPERATION_FAILED</b> <p>The function failed because it was called while the device was in calibration mode.</p> </li> <li>■ <b>TOBII_ERROR_NOT_SUPPORTED</b> <p>The device firmware has no support for setting face type.</p> </li> <li>■ <b>TOBII_ERROR_INTERNAL</b></li> </ul>



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

**See also**      `tobii_get_face_type()` and `tobii_enumerate_face_types()`

## tobii\_get\_face\_type

---

<b>Function</b>	Retreives the current face type setting of the device.
<b>Syntax</b>	<pre>#include &lt;tobii/tobii_advanced.h&gt; tobii_error_t tobii_get_face_type( tobii_device_t* device, tobii_face_type_t* face_type );</pre>
<b>Remarks</b>	<p>A face type is here understood as a class of appearances of the face itself, such as a group of species (e.g. human or crocodile), facial features (e.g. moustache or makeup), or worn objects (e.g. glasses or hats). It is only used for situations where auto detecting such differences is difficult or dangerous.</p> <p><i>device</i> must be a pointer to a valid <code>tobii_device_t</code> instance as created by calling <code>tobii_device_create</code>.</p> <p><i>face_type</i> is a pointer to a zero-terminated string representation of the current face type setting, with a maximum length of 63 characters.</p>
<b>Return value</b>	<p>If the operation is successful, <code>tobii_get_face_type</code> returns <b>TOBII_ERROR_NO_ERROR</b>. If the call fails, <code>tobii_get_face_type</code> returns one of the following:</p> <ul style="list-style-type: none"><li>■ <b>TOBII_ERROR_INVALID_PARAMETER</b> The <i>device</i> or <i>face_type</i> parameter was passed in as NULL.</li><li>■ <b>TOBII_ERROR_CALLBACK_IN_PROGRESS</b> The function failed because it was called from within a callback triggered from an API call such as <code>tobii_device_process_callbacks()</code>, <code>tobii_calibration_retrieve()</code> or <code>tobii_enumerate_illumination_modes()</code>. Calling <code>tobii_get_face_type</code> from within a callback function is not supported.</li><li>■ <b>TOBII_ERROR_INSUFFICIENT_LICENSE</b> The provided license was not a valid consumer level license, or has been blacklisted.</li><li>■ <b>TOBII_ERROR_CONNECTION_FAILED</b> The connection to the device was lost. Call <code>tobii_device_reconnect()</code> to re-establish connection.</li><li>■ <b>TOBII_ERROR_NOT_SUPPORTED</b> The device firmware has no support for retrieving the current face type.</li><li>■ <b>TOBII_ERROR_INTERNAL</b> Some unexpected internal error occurred. This error should normally not be returned, so if it is, please contact the support</li></ul>
<b>See also</b>	<code>tobii_set_face_type()</code> and <code>tobii_enumerate_face_types()</code>