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

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

Thread safety

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

tobii_error_message

Function

Returns a printable error message.

Syntax

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

Remarks

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

Return value

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

Example

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

int main()
{
    tobii_api_t* api;

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

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

    return 0;
}
```

tobii_get_api_version

Function

Query the current version of the API.

Syntax

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

Remarks

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

version is a pointer to an tobii_version_t variable to receive the current version numbers. It contains the following members:

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

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

Example

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

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

    return 0;
}
```

tobii_api_create

Function

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

Syntax

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

Remarks

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

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

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

See also tobii_api_destroy(), tobii_device_create()

Example

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

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

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

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

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

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

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

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

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

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

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

    return 0;
}
```

tobii_api_destroy

Function Destroys an API instance.

Syntax

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

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

tobii_api_destroy should only be called if tobii_api_create completed successfully.

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

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *api* parameter was passed in as NULL.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

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

- **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_create(), tobii_device_destroy()

Example See tobii_api_create()

tobii_enumerate_local_device_urls

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

Syntax

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

Remarks A system might have multiple devices connected, which the stream engine is able to communicate with. tobii_enumerate_local_device_urls iterates over all such (excluding IS1 and IS2) devices found. It will only enumerate devices connected directly to the system, not devices connected on the network. Note that if both a tobii-ttp and a tobii-prp URL is available for the same tracker, only he 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

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

Syntax

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

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

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

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

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

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

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

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

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

- TOBII_DEVICE_GENERATION_G5
- TOBII_DEVICE_GENERATION_IS3
- TOBII_DEVICE_GENERATION_IS4

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.

When creating a connection to a Tobii eye tracker, you need to state whether or not your application is going to store eye tracking data, or transfer eye tracking data. You do this by setting the parameter *field_of_use*, to either TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_TRUE or TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE.

Examples: - If your application will store eye tracking data on the local machine you need to set the parameter to TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_TRUE. - If your application will transfer eye tracking data to a server, to another application, or a cloud system, you need to set the parameter to TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_TRUE. - If your application do not store eye tracking data, but only use it to determine immediate cause of action for your application, you need to set the parameter to TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE.

Please note that eye tracking data refers to both raw eye tracking data, as well as eye tracking data in processed form. This means that if you use the raw eye tracking data and aggregate it or process it into another form, it is still eye tracking data.

The right to store or transfer eye tracking data is governed by Tobiiâ€™s software development licenses. Please see https://developer.tobii.com/license-agreement/ for more information.

For more background information regarding Tobiiâ€™s requirements on storing and transferring eye tracking data, and Tobiiâ€™s Eye Tracking Data Transparency policy, please see https://transparency.tobii.com/.

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_STORE_OR_TRANSFER_FALSE**

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. (Deprecated name: TOBII_FIELD_OF_USE_INTERACTIVE)

- **TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_TRUE**

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. (Deprecated name: TOBII_FIELD_OF_USE_ANALYTICAL)

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_CONNECTION_FAILED**

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

- **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_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.

- **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.

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 <string.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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

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

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

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

tobii_device_destroy

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

Syntax

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

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

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

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

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

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

- **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_device_create_ex()

Example See tobii_device_create()

tobii_wait_for_callbacks

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

Syntax

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

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

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

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

- **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_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 <string.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_STORE_OR_TRANSFER_FALSE, &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

Function

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

Syntax

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

Remarks

Stream engine does not do any kind of background processing, it doesn't start any threads. It doesn't use any asynchronous

callbacks. This means that in order to receive data from the device, the application needs to manually request the callbacks to happen synchronously, and this is done by calling tobii_device_process_callbacks.

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

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

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

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

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

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **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_CALLBACK_IN_PROGRESS**

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

- **TOBII_ERROR_INTERNAL**

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

See also

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

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

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

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

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

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

    return 0;
```

```
}
```

tobii_device_clear_callback_buffers

Function Removes all unprocessed entries from the callback queues.

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

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

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

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

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

See also tobii_wait_for_callbacks(), tobii_device_process_callbacks()

tobii_device_reconnect

Function Establish a new connection after a disconnect.

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

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

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

Return value

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_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.

- **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.

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_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **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_INTERNAL**

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

See also

tobii_wait_for_callbacks(), tobii_device_reconnect(), tobii_device_process_callbacks(), tobii_system_clock()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &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

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

Function

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

Syntax `tobii_error_t tobii_system_clock( tobii_api_t* api, int64_t* timestamp_us );`

Remarks

Many of the data streams provided by the stream engine API, contains a timestamp value, measured in microseconds (us). The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values. To facilitate making comparisons between stream engine provided timestamps and application specific events, tobii_system_clock can be used to retrieve a timestamp using the same clock and same relative values as the timestamps used in stream engine callbacks.

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

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

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

See also

tobii_api_create()

Example

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

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

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

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

    return 0;
}
```

tobii_get_device_info

Function

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

Syntax

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

Remarks

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

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

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

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_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.

- **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 <string.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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

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

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

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

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

tobii_get_track_box

Function

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

Syntax

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

Remarks

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

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

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

- *front_upper_right_xyz*, *front_upper_left_xyz*, *front_lower_left_xyz*, *front_lower_right_xyz*

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

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

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

This function is not supported on all Tobii devices, please contact Tobii support if you have questions.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_CALLBACK_IN_PROGRESS**

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

- **TOBII_ERROR_INTERNAL**

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

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &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_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **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.

- **TOBII_ERROR_INTERNAL**

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

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

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

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

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_get_state_uint32

Function

Gets the current value of a state in the tracker.

Syntax

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

Remarks

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

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

- **TOBII_STATE_CALIBRATION_ID**

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

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

NOTE: This method relies on cached values which is updated when tobii_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 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_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **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.

- **TOBII_ERROR_INTERNAL**

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

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

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

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

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_get_state_string

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

Syntax

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

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

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

- **TOBII_STATE_FAULT**

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

- **TOBII_STATE_WARNING**

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

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

NOTE: This method relies on cached values which is updated when tobii_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 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 <string.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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

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

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

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_capability_supported

Function

Ask if a specific feature is supported or not.

Syntax

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

Remarks

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

capability is one of the enum values in tobii_capability_t:

- **TOBII_CAPABILITY_DISPLAY_AREA_WRITABLE**

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

- **TOBII_CAPABILITY_CALIBRATION_2D**

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

- **TOBII_CAPABILITY_CALIBRATION_3D**

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

- **TOBII_CAPABILITY_PERSISTENT_STORAGE**

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

- **TOBII_CAPABILITY_CALIBRATION_PER_EYE**

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

- **TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_3D_GAZE_COMBINED**

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

- **TOBII_CAPABILITY_FACE_TYPE**

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

- **TOBII_CAPABILITY_COMPOUND_STREAM_USER_POSITION_GUIDE_XY**

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

- **TOBII_CAPABILITY_COMPOUND_STREAM_USER_POSITION_GUIDE_Z**

  Query if the device supports the z-coordinate of the user position guide stream.

- **TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_LIMITED_IMAGE**

  Query if the device supports the wearable limited image stream.

- **TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_PUPIL_DIAMETER**

  Query if the device supports pupil diamater in the wearable data stream.

- **TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_PUPIL_POSITION**

  Query if the device supports pupil position in the wearable data stream.

- **TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_EYE_OPENNESS**

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

- **TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_3D_GAZE_PER_EYE**

  Query if the device supports per eye 3D gaze in the wearable data stream.

- **TOBII_CAPABILITY_COMPOUND_STREAM_WEARABLE_USER_POSITION_GUIDE_XY**

  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_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.

- **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_stream_supported()

Example

```
#include <tobii/tobii.h>
#include <stdio.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &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_GAZE_DATA**
- **TOBII_STREAM_DIGITAL_SYNCPORT**
- **TOBII_STREAM_DIAGNOSTICS_IMAGE**
- **TOBII_STREAM_USER_POSITION_GUIDE**
- **TOBII_STREAM_WEARABLE_CONSUMER**
- **TOBII_STREAM_WEARABLE_ADVANCED**
- **TOBII_STREAM_WEARABLE_FOVEATED_GAZE**
- **TOBII_STREAM_RESPONSIVE_GAZE_POINT**

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

```c
#include <tobii/tobii.h>
#include <stdio.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

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

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

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_streams.h

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

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

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

tobii_gaze_point_subscribe

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

Syntax

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

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

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

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

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

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

- *gaze_point*

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

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

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

Return value If the operation is successful, tobii_gaze_point_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_gaze_point_subscribe returns an error code specific to the device.

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

Example

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

void gaze_point_callback( tobii_gaze_point_t const* gaze_point, void* user_data )
{
    (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_STORE_OR_TRANSFER_FALSE, &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

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

Syntax

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

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

Return value If the operation is successful, tobii_gaze_point_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_gaze_point_unsubscribe returns an error code specific to the device.

See also tobii_gaze_point_subscribe()

Example See tobii_gaze_point_subscribe()

tobii_gaze_origin_subscribe

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

Syntax

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

Remarks This subscription is for receiving the origin of the gaze vector, measured in millimeters from the center of the display. Gaze origin is a point on the users eye, but the exact point of the origin varies by device. For example, it might be defined as the center of the pupil or the center of the cornea. The data is lightly filtered for stability.

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

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

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

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

- *gaze_origin*

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

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

- *right_xyz* An array of three floats, for the x, y and z coordinate of the gaze origin point on the right eye of the user, as measured in millimeters from the center of the display.

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

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

Return value If the operation is successful, tobii_gaze_origin_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_gaze_origin_subscribe returns 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 <string.h>
#include <assert.h>

void gaze_origin_callback( tobii_gaze_origin_t const* gaze_origin, void* user_data )
{
    (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_STORE_OR_TRANSFER_FALSE, &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

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

Syntax

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

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

Return value If the operation is successful, tobii_gaze_origin_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails,

tobii_gaze_origin_unsubscribe returns an error code specific to the device.

See also tobii_gaze_origin_subscribe()

Example See tobii_gaze_origin_subscribe()

tobii_eye_position_normalized_subscribe

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

Syntax

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

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

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

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

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

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

- *eye_position*

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

  - *timestamp_us*

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

  - *left_validity*

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

  - *left_xyz*

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

  - *right_validity*

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

  - *right_xyz*

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

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

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

This function is not supported on all Tobii devices. In general we suggest using tobii_user_position_guide_subscribe() over tobii_eye_position_normalized_subscribe(). Please contact Tobii support if you have questions.

Return value If the operation is successful, tobii_eye_position_normalized_subscribe returns **TOBII_ERROR_NO_ERROR**. Returns **TOBII_ERROR_NOT_SUPPORTED** when not available. 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 <string.h>
#include <assert.h>

void eye_position_callback( tobii_eye_position_normalized_t const* eye_pos, void* user_data )
{
    (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_STORE_OR_TRANSFER_FALSE, &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

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

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

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

Return value If the operation is successful, tobii_eye_position_normalized_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_eye_position_normalized_unsubscribe returns an error code specific to the device.

See also tobii_eye_position_normalized_subscribe()

Example See tobii_eye_position_normalized_subscribe()

tobii_user_presence_subscribe

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

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

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

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

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

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

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

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

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

Return value If the operation is successful, tobii_user_presence_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_user_presence_subscribe returns an error code specific to the device.

See also tobii_user_presence_unsubscribe(), tobii_device_process_callbacks(), tobii_system_clock()

Example

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

void presence_callback( tobii_user_presence_status_t status, int64_t timestamp_us, void* user_data )
{
    (void)user_data;
    (void)timestamp_us;
    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_STORE_OR_TRANSFER_FALSE, &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

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

Syntax

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

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

Return value If the operation is successful, tobii_user_presence_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_user_presence_unsubscribe returns an error code specific to the device.

See also tobii_user_presence_subscribe()

Example See tobii_user_presence_subscribe()

tobii_head_pose_subscribe

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

Syntax

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

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

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

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

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

- *head_pose*

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

  - *timestamp_us*

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

  - *position_validity*

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

  - *position_xyz*

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

  - *rotation_validity_xyz*

    An array indicating the validity of each elemnt of the rotation_xyz field. **TOBII_VALIDITY_INVALID** if the element is not valid, **TOBII_VALIDITY_VALID** if it is.

  - *rotation_xyz*

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

- *user_data*

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

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

Return value If the operation is successful, tobii_head_pose_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_head_pose_subscribe returns an error code specific to the device.

See also tobii_head_pose_unsubscribe()

Example

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

void head_pose_callback( tobii_head_pose_t const* head_pose, void* user_data )
{
    (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_STORE_OR_TRANSFER_FALSE, &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

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

Syntax

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

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

Return value If the operation is successful, tobii_head_pose_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_head_pose_unsubscribe returns an error code specific to the device.

See also tobii_head_pose_subscribe()

Example See tobii_head_pose_subscribe()

tobii_notifications_subscribe

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

Syntax

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

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

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

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

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

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

- *notification*

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

- *type*

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

  TOBII_NOTIFICATION_TYPE_CALIBRATION_STATE_CHANGED
  TOBII_NOTIFICATION_TYPE_EXCLUSIVE_MODE_STATE_CHANGED
  TOBII_NOTIFICATION_TYPE_TRACK_BOX_CHANGED TOBII_NOTIFICATION_TYPE_DISPLAY_AREA_CHANGED
  TOBII_NOTIFICATION_TYPE_FRAMERATE_CHANGED
  TOBII_NOTIFICATION_TYPE_POWER_SAVE_STATE_CHANGED
  TOBII_NOTIFICATION_TYPE_DEVICE_PAUSED_STATE_CHANGED
  TOBII_NOTIFICATION_TYPE_CALIBRATION_ENABLED_EYE_CHANGED
  TOBII_NOTIFICATION_TYPE_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 unmodified when registering the callback.

The different possible faults and warnings returned by TOBII_NOTIFICATION_TYPE_FAULTS_CHANGED and TOBII_NOTIFICATION_TYPE_WARNINGS_CHANGED are described below

Fault message	Description
Memory	Memory allocation error
etp	Internal error in libetp
config	Invalid configuration (e.g. a required key is missing)
transport	Error in transport layer
algo	Error in Algobox or Algo thread
general-tdi	General hardware or TDI thread error
access	Unable to get access to a needed resource
camera-driver	Camera driver error
camera-0	Error with first camera
camera-1	Error with second camera
led-driver	Illuminator driver error
led-0	Error with LED 0
led-1	Error with LED 1
led-2	Error with LED 2
led-3	Error with LED 3
remote-system	Cannot communicate with remote system (e.g. EyeChip or a peripheral device)
extctrl-0	Error with external controller 0
extctrl-1	Error with external controller 1
cpu-temperature-critical	CPU overheated
camera0-temperature-critical	Camera 0 overheated
camera1-temperature-critical	Camera 1 overheated
led0-temperature-critical	LED 0 overheated
led1-temperature-critical	LED 1 overheated
led2-temperature-critical	LED 2 overheated
led3-temperature-critical	LED 3 overheated
system-temperature-critical	Overall system overheated
system-current-high	Too high current draw

Warning message	Description
cpu-temperature-high	CPU nearly overheated
camera0-temperature-high	Camera 0 nearly overheated
camera1-temperature-high	Camera 1 nearly overheated
led0-temperature-high	LED 0 nearly overheated
led1-temperature-high	LED 1 nearly overheated
led2-temperature-high	LED 2 nearly overheated
led3-temperature-high	LED 3 nearly overheated
system-temperature-high	Overall system nearly overheated
camera-bad-calib	Bad camera calibration
cpu-fan	Fan stuck or too slow
system-voltage-low	Supply voltage too low
system-voltage-high	Supply voltage too high
system-power-high	Power usage suspiciously high
usb-power-capability-low	USB bus low power capability

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 <string.h>
#include <assert.h>

void notifications_callback( tobii_notification_t const* notification, void* user_data )
{
    (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_STORE_OR_TRANSFER_FALSE, &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;
}
```

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

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

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

Return value If the operation is successful, tobii_notifications_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_notifications_unsubscribe returns an error code specific to the device.

See also tobii_notifications_subscribe()

Example See tobii_notifications_subscribe()

tobii_user_position_guide_subscribe

Function Start listening to the user position guide stream, which is used to help a user position their eyes in the track box correctly.

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

Remarks The user position guide stream is used to help users position their eyes correctly in the track box frustum of a remote eye tracker. It gives the current position of the left and right eye in normalized 3d-coordinates.

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 user_position_guide_callback(tobii_user_position_guide_t const * user_position_guide, void* user_data);

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:

- *user_position_guide*
  - *timestamp_us*

    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, **TOBI_VALIDITY_VALID** if it is.
  - *left_position_normalized_xyz*

    Floating point array containing the normalized 3d-coordinate of the left eye in the range 0.0 to 1.0.
  - *right_position_validity*

    Indicates the validity of the right_position_xyz field. **TOBII_VALIDITY_INVALID** if the field is not valid, **TOBI_VALIDITY_VALID** if it is.
  - *right_position_normalized_xyz*

    Floating point array containing the normalized 3d-coordinate of the right eye in the range 0.0 to 1.0.
- *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 <string.h>
#include <assert.h>

void user_position_guide_callback( tobii_user_position_guide_t const* position_guide, void* user_data )
{
    (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( "Left 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_STORE_OR_TRANSFER_FALSE, &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

Function Stops listening to the user position guide that was subscribed to by a call to tobii_user_position_guide_subscribe().

Syntax

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

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

Return value If the operation is successful, tobii_user_position_guide_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_user_position_guide_unsubscribe returns an error code specific to the device.

See also tobii_user_position_guide_subscribe()

Example See tobii_user_position_guide_subscribe()

tobii_responsive_gaze_point_subscribe

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

Syntax

```
#include <tobii/tobii_streams.h>
tobii_error_t tobii_responsive_gaze_point_subscribe( tobii_device_t* device,
    tobii_responsive_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 not filtered for responsiveness.

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 responsive_gaze_point_callback( tobii_responsive_gaze_point_t const* gaze_point, void* user_data )
```

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

- *gaze_point*

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

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

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

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

Return value If the operation is successful, tobii_responsive_gaze_point_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_responsive_gaze_point_subscribe returns an error code specific to the device.

See also tobii_responsive_gaze_point_unsubscribe(), tobii_device_process_callbacks(), tobii_system_clock()

Example

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

void responsive_gaze_point_callback( tobii_responsive_gaze_point_t const* gaze_point, void* user_data )
{
    (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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    error = tobii_responsive_gaze_point_subscribe( device, responsive_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_responsive_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_responsive_gaze_point_unsubscribe

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

Syntax

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

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

Return value If the operation is successful, tobii_responsive_gaze_point_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails,

tobii_responsive_gaze_point_unsubscribe returns an error code specific to the device.

See also tobii_responsive_gaze_point_subscribe()

Example See tobii_responsive_gaze_point_subscribe()

tobii_wearable.h

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

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

tobii_wearable_consumer_data_subscribe

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

Syntax

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

Remarks All coordinates are expressed in a right-handed Cartesian system with Z facing forward from the eye.

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

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

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

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

- *data* This is a pointer to a struct containing the data listed below. Note that it is only valid during the callback. Its data should be copied if access is necessary at a later stage, from outside the callback.
  - *timestamp_us* Timestamp value for when the data was captured, measured in microseconds (us), and synchronized with the clock of the computer. The function tobii_system_clock can be used to retrieve a timestamp (at the time of the call) using the same clock and same relative values as this timestamp. The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values.
  - *left* This is a struct containing the following data, related to the left eye:
    - *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_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_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_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_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.

- **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.

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 )
{
    (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_STORE_OR_TRANSFER_FALSE, &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>
tobii_error_t TOBII_CALL tobii_wearable_consumer_data_unsubscribe( tobii_device_t* device );
```

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

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **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_NOT_SUBSCRIBED**

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

- **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_unsubscribe from within a callback function is not supported.

- **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_consumer_data_subscribe()

tobii_wearable_advanced_data_subscribe

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

Syntax

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

```
tobii_error_t TOBII_CALL tobii_wearable_advanced_data_subscribe( tobii_device_t* device,
    tobii_wearable_advanced_data_callback_t callback, void* user_data );
```

Remarks

All coordinates are expressed in a right-handed Cartesian system with Z facing forward from the eye.

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

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

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

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

- *data* This is a pointer to a struct containing the data listed below. Note that it is only valid during the callback. Its data should be copied if access is necessary at a later stage, from outside the callback.
  - *timestamp_tracker_us* Timestamp value for when the 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 data was captured, measured in microseconds (us), and synchronized with the clock of the computer. The function tobii_system_clock can be used to retrieve a timestamp (at the time of the call) using the same clock and same relative values as this timestamp. The epoch is undefined, so these timestamps are only useful for calculating the time elapsed between a pair of values.
  - *left* This is a struct containing the following data, related to the left eye:
    - *gaze_origin_validity* **TOBII_VALIDITY_INVALID** if *gaze_origin_mm_xyz* is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.
    - *gaze_origin_mm_xyz* An array of three floats, for the x, y and z coordinate of the point in the user's eye from which the calculated gaze ray originates, expressed in a right-handed Cartesian coordinate system. See the wearable hardware specification for its origin.
    - *gaze_direction_validity* **TOBII_VALIDITY_INVALID** if *gaze_direction_normalized_xyz* for the eye is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.
    - *gaze_direction_normalized_xyz* An array of three floats, for the x, y and z coordinate of the gaze direction of the eye of the user, expressed as a unit vector in a right-handed Cartesian coordinate system.
    - *pupil_diameter_validity* **TOBII_VALIDITY_INVALID** if *pupil_diameter_mm* is not valid for this frame, **TOBII_VALIDITY_VALID** if it is.
    - *pupil_diameter_mm* A float that represents the approximate diameter of the pupil, expressed in millimeters. Only relative changes are guaranteed to be accurate.
    - *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_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_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_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.

- **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.

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 )
{
    (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_STORE_OR_TRANSFER_FALSE, &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

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

Syntax

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

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

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **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_NOT_SUBSCRIBED**

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

- **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.

- **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_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_NOT_SUPPORTED**

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

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_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.

- **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_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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

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

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

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

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

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

tobii_set_lens_configuration

Function Sets the current lens configuration in the tracker.

Syntax

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

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

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

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

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

  The provided license does not permit this operation.

- **TOBII_ERROR_NOT_SUPPORTED**

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

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_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.

- **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_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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );
```

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

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

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

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

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


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

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

tobii_lens_configuration_writable

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

Syntax

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

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

writable must be a pointer to a valid tobii_lens_configuration_writable_t.

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

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_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.

- **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_lens_configuration(), tobii_set_lens_configuration()

tobii_get_lens_configuration_extended

Function Retrieves the current lens configuration in the tracker.

Syntax

```
#include <tobii/tobii_wearable.h>
tobii_error_t TOBII_CALL tobii_get_lens_configuration_extended( tobii_device_t* device,
    tobii_lens_configuration_extended_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_extended_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* This is a struct containing the following data, related to the left eye:
  - *offset* An array of three floats, for the x, y and z offset of the left lens in the headset, given in millimeters.
  - *rotation* An array of three floats, for the x, y and z intrinsic rotation of the left lens in the headset, given in degrees.
- *right* This is another instance of the same struct as in *left*, but which holds data related to the right eye of the user.

The offset and the rotation are expressed in a right handed Cartesian coordinate system where: X-axis positive direction is to the left when facing out through the lens. Y-axis positive direction is towards the upper part of the HMD. Z-axis positive direction is from the eye of the user towards the eye-tracker.

X rotation is a clockwise angle around the positive direction of the x-axis, known as pitch. Y rotation is a clockwise angle around the positive direction of the y-axis, known as yaw. Z rotation is a clockwise angle around positive direction of the z-axis, known as roll.

Rotations are intrinsic and computed in x-y-z order.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_NOT_SUPPORTED**

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

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_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_extended from within a callback function is not supported.

- **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_lens_configuration_extended()

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

    tobii_lens_configuration_extended_t lens_config;
    error = tobii_get_lens_configuration_extended( device, &lens_config );
    assert( error == TOBII_ERROR_NO_ERROR );

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

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

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

    printf( "VR lens rotation [deg] (right): (%f, %f, %f)\n",
```

```
        lens_config.right.intrinsic_rotation_deg_xyz[ 0 ],
        lens_config.right.intrinsic_rotation_deg_xyz[ 1 ],
        lens_config.right.intrinsic_rotation_deg_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_extended

Function

Sets the current lens configuration in the tracker.

Syntax

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

Remarks

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

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

- *left* This is a struct containing the following data, related to the left eye:
  - *offset* An array of three floats, for the x, y and z offset of the left lens in the headset, given in millimeters.
  - *rotation* An array of three floats, for the x, y and z intrinsic rotation of the left lens in the headset, given in degrees.
- *right* This is another instance of the same struct as in *left*, but which holds data related to the right eye of the user.

The offset and the rotation are expressed in a right handed Cartesian coordinate system where: X-axis positive direction is to the left when facing out through the lens. Y-axis positive direction is towards the upper part of the HMD. Z-axis positive direction is from the eye of the user towards the eye-tracker.

X rotation is a clockwise angle around the positive direction of the x-axis, known as pitch. Y rotation is a clockwise angle around the positive direction of the y-axis, known as yaw. Z rotation is a clockwise angle around positive direction of the z-axis, known as roll.

Rotations are intrinsic and computed in x-y-z order.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

  The provided license does not permit this operation.

- **TOBII_ERROR_NOT_SUPPORTED**

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

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_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_extended from within a callback function is not supported.

- **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_lens_configuration_extended()

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

    tobii_lens_configuration_extended_writable_t writable;
    error = tobii_lens_configuration_extended_writable( device, &writable );
    assert( error == TOBII_ERROR_NO_ERROR );

    if( writable == TOBII_LENS_CONFIGURATION_EXTENDED_WRITABLE )
    {
        tobii_lens_configuration_extended_t lens_config;

        //Add 32 mm offset for each lens on the X-axis
        lens_config.left.offset_mm_xyz[ 0 ] = 32.0;
        lens_config.right.offset_mm_xyz[ 0 ] = -32.0;

        lens_config.left.offset_mm_xyz[ 1 ] = 0.0;
        lens_config.right.offset_mm_xyz[ 1 ] = 0.0;

        lens_config.left.offset_mm_xyz[ 2 ] = 0.0;
        lens_config.right.offset_mm_xyz[ 2 ] = 0.0;

        //Add 5 degrees tilt for each module (rotation around the X-axis)
        lens_config.left.intrinsic_rotation_deg_xyz[ 0 ] = 5.0;
        lens_config.right.intrinsic_rotation_deg_xyz[ 0 ] = -5.0;

        //Add 3 degrees wrap for each module (rotation around the y-axis)
        lens_config.left.intrinsic_rotation_deg_xyz[ 1 ] = 3.0;
        lens_config.right.intrinsic_rotation_deg_xyz[ 1 ] = -3.0;

        lens_config.left.intrinsic_rotation_deg_xyz[ 1 ] = 0.0;
        lens_config.right.intrinsic_rotation_deg_xyz[ 1 ] = 0.0;

        error = tobii_set_lens_configuration_extended( device, &lens_config );
        assert( error == TOBII_ERROR_NO_ERROR );
    }
    else
        printf( "Unable to write extended 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_extended_writable

Function

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

Syntax

```
#include <tobii/tobii_wearable.h>
tobii_error_t TOBII_CALL tobii_lens_configuration_extended_writable( tobii_device_t* device,
    tobii_lens_configuration_extended_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_extended_writable_t.

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

Return value

If the operation is successful, tobii_lens_configuration_extended_writable() returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_lens_configuration_extended_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_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_extended_writable from within a callback function is

not supported.

- **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_lens_configuration_extended(), tobii_set_lens_configuration_extended()

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 fovaeted 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 blocklisted.

- **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_STORE_OR_TRANSFER_FALSE, &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;
}
```

tobii_wearable_foveated_gaze_unsubscribe

Function

Stops listening to wearable foveated gaze stream that was subscribed to by a call to tobii_wearable_foveated_gaze_unsubscribe()

Syntax

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

Remarks

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

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL. *device* must be a valid tobii_device_t pointer as created by tobii_device_create.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUBSCRIBED**

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

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

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

Syntax

```
#include <tobii/tobii.h>
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 );
```

Remarks 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.

When creating a connection to a Tobii eye tracker, you need to state whether or not your application is going to store eye tracking data, or transfer eye tracking data. You do this by setting the parameter *field_of_use*, to either TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_TRUE or TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE.

Examples: - If your application will store eye tracking data on the local machine you need to set the parameter to TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_TRUE. - If your application will transfer eye tracking data to a server, to another application, or a cloud system, you need to set the parameter to TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_TRUE. - If your application do not store eye tracking data, but only use it to determine immediate cause of action for your application, you need to set the parameter to TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE.

Please note that eye tracking data refers to both raw eye tracking data, as well as eye tracking data in processed form. This means that if you use the raw eye tracking data and aggregate it or process it into another form, it is still eye tracking data.

The right to store or transfer eye tracking data is governed by Tobiiâ€™s software development licenses. Please see https://developer.tobii.com/license-agreement/ for more information.

For more background information regarding Tobiiâ€™s requirements on storing and transferring eye tracking data, and Tobiiâ€™s Eye Tracking Data Transparency policy, please see https://transparency.tobii.com/.

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_STORE_OR_TRANSFER_FALSE**

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. (Deprecated name: TOBII_FIELD_OF_USE_INTERACTIVE)

- **TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_TRUE**

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. (Deprecated name: TOBII_FIELD_OF_USE_ANALYTICAL)

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 should be provided. It is an array for returning the results of the license validation for each license. It is adviced 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, it's 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_CONNECTION_FAILED**

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

- **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_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.

- **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.

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.

- **TOBII_LICENSE_VALIDATION_RESULT_REVOKED**

The license has been revoked.

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 <stdlib.h>
#include <memory.h>
#include <assert.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &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

Function Stores the license key on the tracker

Syntax

```
#include <tobii/tobii.h>
tobii_error_t tobii_license_key_store( tobii_device_t* device, void* data, size_t size );
```

Remarks 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.

device must be a pointer to a variable of the type tobii_device_t* that is, a pointer to a tobii_device_t.

data has to be in uint16_t text passed as the void*. It is optional and hence if it is 0 then it will erase already stored license

size is the no of bytes in the data buffer. If it is passed as 0 then it will erase already stored license.

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 *device* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUPPORTED**

  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.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **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_OPERATION_FAILED**

  Writting to the the device failed because of unexpected IO error, file not found, storage is full or filename is invalid.

- **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_license_key_store from within a callback function is not supported.

- **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_license_key_retrieve(), tobii_device_create()

Example

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

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <assert.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &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

Function Retreives the already stored license key from the device.

Syntax

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

Remarks 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.

device must be a pointer to a variable of the type tobii_device_t* that is, a pointer to a tobii_device_t-pointer. the device is obtained by calling tobii_device_create() or by tobii_device_create_ex(). It must be freed by calling tobii_device_destroy() as clean up operation.

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

```
void data_receiver( void const* data, size_t size, void* user_data )
```

This function will be called with the retrieved license data. It is called with the following parameters:

- *data* 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.
- *size* This gives the size of the data buffer read.
- *user_data* This is the custom pointer sent in to tobii_license_key_retrieve.

user_data is optional. Caller can pass any data here as the calling device which could be used in the *receiver*.

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 *device* parameter was passed in as NULL.

- **TOBII_ERROR_NOT_SUPPORTED**

  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.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **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_OPERATION_FAILED**

  Reading from the device failed because of unexpected IO error, file not found, filename is invalid.

- **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_license_key_retrieve from within a callback function is not supported.

- **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_license_key_retrieve(), tobii_device_create()

Example

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

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <assert.h>
#include <string.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_STORE_OR_TRANSFER_FALSE, &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

Function Retrieves the currently active feature group for a device.

Syntax

```
#include <tobii/tobii_advanced.h>
tobii_error_t tobii_get_feature_group( tobii_device_t* device, tobii_feature_group_t* feature_group );
```

Remarks The currently active feature group is determined by tobii_device_create based on the license key passed into it. tobii_get_feature_group can be used to query the currently active feature group.

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

feature_group is a pointer to a tobii_feature_group_t to receive the current group, in the form of values from the following enum:

- **TOBII_FEATURE_GROUP_BLOCKED**

  The provided license key was invalid, or the application making the call has been blocklisted. No API functionality will be available.

- **TOBII_FEATURE_GROUP_CONSUMER**

  Default feature group for passing a NULL (default) license key to tobii_device_create. Gives access to all API functions except those where a higher feature group is specified in the documentation.

- **TOBII_FEATURE_GROUP_CONFIG**

  Grants access to functionality that changes configuration of the tracker (mainly in tobii_config.h). This feature group might be automatically granted for certain devices, like head-mounted displays, even if a default license key is used.

- **TOBII_FEATURE_GROUP_PROFESSIONAL**

  Gives access to the functionality in tobii_advanced.h. This feature group might be automatically granted for professional level devices, as supplied by Tobii Pro, even if a default license key is used.

- **TOBII_FEATURE_GROUP_INTERNAL**

  For internal use by Tobii.

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 **TOBII_FEATURE_GROUP_CONSUMER** is required).

Each feature group includes all feature groups preceding it (with the exception of **TOBII_FEATURE_GROUP_BLOCKED**, which indicates that the specified license key was found to be invalid, or the current application has been blocklisted, in which case no API functions will be available).

Return value

If the feature group was successfully retrieved, tobii_get_feature_group returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_get_feature_group returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  One or more of the *device* and *feature_group* parameters were passed in as NULL. *device* must be a valid tobii_device_t pointer as created by tobii_device_create, and *feature_group* must be a valid pointer to a tobii_feature_group_t variable.

- **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_feature_group from within a callback function is not supported.

- **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()

Example

```
#include <tobii/tobii_licensing.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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_feature_group_t feature_group;
    error = tobii_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 = tobii_device_destroy( device );
    assert( error == TOBII_ERROR_NO_ERROR );

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

tobii_config.h

The tobii_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 tobii_device_create_ex.

tobii_set_enabled_eye

Function

Controls which eyes are used when calculating the combined gaze.

Syntax

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

Remarks

Controls which eyes are used when calculating the combined gaze point in the gaze point stream.

If for example the eye tracker user has a visual impairment in one eye or suffer from an eye disorder such as strabismus, then it would be beneficial to only enable the dominant eye.

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 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, tobii_set_enabled_eye returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_set_enabled_eye returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter passed was NULL.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

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

- **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_output_frequency from within a callback function is not supported.

- **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_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
};

    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_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

Queries which eyes are used when calculating the combined gaze point in the gaze point stream.

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

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_enabled_eye 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_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

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

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **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_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()
    // and on how to create/destroy a device using tobii_device_create/destroy().
    tobii_device_t* device = 0;

    tobii_enabled_eye_t enabled_eye = TOBII_ENABLED_EYE_BOTH;
    tobii_error_t error = tobii_get_enabled_eye( device, &enabled_eye );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Failed to get enabled eye property\n" );
        // destroy api and device, omitted here for brevity
        return error;
    }

    switch( enabled_eye )
    {
        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;
    }

    // destroy api and device, omitted here for brevity
    return 0;
}
```

tobii_calibration_start

Function Starts the calibration process, placing the tracker in a state ready to receive data collection requests.

Syntax

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

Remarks If this call is successful, the client receives exclusive access to the calibration process, no other clients can start a calibration process until tobii_calibration_stop() has been called.

These functions requires that tobii_calibration_start() has been called successfully:

- 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()

Here follows a basic calibration process example where a 7-point calibration is performed, it's recommended to do a 7-point calibration in-order to get good accuracy of the calibration:

- Call tobii_calibration_start() to initiate the calibration process
- Start by doing a one point calibration at center screen
  - Plot a stimuli point in UI and subscribe to gaze point to check when user is in the vicinity of the stimuli point.
  - Once user has directed its gaze to the stimuli point, call tobii_calibration_collect_data_X().
  - Call tobii_calibration_compute_and_apply() to commit the one point calibration.
- Improve the calibration by doing a three point calibration
  - Plot stimuli points in a specific pattern in UI and check when user is in the vicinity of a one of the stimuli points.
  - Once user has directed its gaze to a specific the stimuli point, call tobii_calibration_collect_data_X() for that stimuli point, repeat for each stimuli point.
  - If a call to tobii_calibration_collect_data_X() should fail, re-run it until a successful collection is achieved.
  - Once all points have been successfully collected call tobii_calibration_compute_and_apply() to commit the three point calibration.
- Improve the calibration further by doing another round of three point calibration
  - Plot stimuli points inverting the pattern from previous step, and repeat the above steps for each point.
  - Call tobii_calibration_compute_and_apply() to commit the last three point calibration.
- End the calibration process by calling tobii_calibration_stop(), this stops the exclusive calibration access.

For a more detailed calibration process description please consult: https://developer.tobii.com/download-packages/how-to-create-a-customized-calibration

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

DEPRECATED enabled_eye must ALWAYS be **TOBII_ENABLED_EYE_BOTH**, this parameter is deprecated and will be removed in the next major release.

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  A value other than TOBII_ENABLED_EYE_BOTH was passed for the *enabled_eye* parameter.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_CALIBRATION_ALREADY_STARTED**

  tobii_calibration_start has already been called, and not yet been stopped by calling tobii_calibration_stop. A started calibration must always be stopped before a new calibration is started.

- **TOBII_ERROR_CALIBRATION_BUSY**

  Another client is already calibrating the device. Only one calibration can be running at a time, across all connected clients.

- **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_start from within a callback function is not supported.

- **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_stop(), tobii_calibration_clear(), 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_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example See tobii_calibration_collect_data_2d().

tobii_calibration_stop

Function Signals that the calibration process has been completed and that no further data collection will be requested.

Syntax

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

Remarks This function should not be called unless the calibration process has been started by calling tobii_calibration_start().

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_stop returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_calibration_stop returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **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_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_stop from within a callback function is not supported.

- **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_clear(), 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_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example See tobii_calibration_collect_data_2d()).

tobii_calibration_collect_data_2d

Function Performs data collection for the specified screen coordinate.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_calibration_collect_data_2d( tobii_device_t* device,
    float x, float y );
```

Remarks This function should not be called unless the calibration process has been started by calling tobii_calibration_start().

This function is synchronous and may block for several seconds while the eye tracker collects calibration data.

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 normalized (0 to 1) coordinates.

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

Return value If the operation is successful, tobii_calibration_collect_data_2d returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_calibration_collect_data_2d returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **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_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_2d from within a callback function is not supported.

- **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_clear(), 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_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example

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

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <assert.h>
#include <string.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!\n" );
        return 0;
    }

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

    if( file_size <= 0 )
    {
        fclose( license_file );
        printf( "Unable to read License file!\n" );
        return 0;
    }

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

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

tobii_error_t create_device( tobii_api_t* api, tobii_device_t** device, char* url )
{
    tobii_error_t 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 license_result;
    error = tobii_device_create_ex( api, url, TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE, &license, 1, &license_result,
device );
    free( license_key );
    license_key = 0;

    if( error == TOBII_ERROR_CONNECTION_FAILED )
    {
        printf( "Failed to connect to tracker.\n" );
        return error;
    }
```

```
    return error;
}

tobii_error_t calibrate( tobii_device_t* device )
{
    tobii_error_t error = tobii_calibration_start( device, TOBII_ENABLED_EYE_BOTH );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    printf( "Look at the top left!\n" );
    error = tobii_calibration_collect_data_2d( device, 0.1f, 0.1f );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    printf( "Look at the bottom right!\n" );
    error = tobii_calibration_collect_data_2d( device, 0.9f, 0.9f );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    printf( "Look at the center!\n" );
    error = tobii_calibration_collect_data_2d( device, 0.5f, 0.5f );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    error = tobii_calibration_compute_and_apply( device );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    return tobii_calibration_stop( device );
}

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 = create_device( api, &device, url );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Failed to create device!\n" );
        tobii_api_destroy( api );
        return error;
    }

    error = calibrate( device );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "Calibration succeed!\n" );
    }
    else
    {
        printf( "Calibration failed!\n" );
    }

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_calibration_collect_data_3d

Function Performs data collection for the specified 3d coordinate.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_calibration_collect_data_3d( tobii_device_t* device,
    float x, float y, float z );
```

Remarks This function should not be called unless the calibration process has been started by calling tobii_calibration_start().

This function is synchronous and may block for several seconds while the eye tracker collects calibration data.

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_INSUFFICIENT_LICENSE**

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

- **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_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_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_clear(), tobii_calibration_collect_data_2d(), 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_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example

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

#include <stdio.h>
#include <stdlib.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!\n" );
        return 0;
    }

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

    if( file_size <= 0 )
    {
        fclose( license_file );
        printf( "Unable to read License file!\n" );
        return 0;
    }

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

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

tobii_error_t create_device( tobii_api_t* api, tobii_device_t** device, char* url )
{
    tobii_error_t 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 license_result;
    error = tobii_device_create_ex( api, url, TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE, &license, 1, &license_result,
device );
    free( license_key );
    license_key = 0;

    if( error == TOBII_ERROR_CONNECTION_FAILED )
    {
```

```
        printf( "Failed to connect to tracker.\n" );
        return error;
    }

    return error;
}

tobii_error_t calibrate( tobii_device_t* device )
{
    tobii_error_t error = tobii_calibration_start( device, TOBII_ENABLED_EYE_BOTH );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    // plot an object at (0.1, 0.1, 0.1) that the user will look at
    error = tobii_calibration_collect_data_3d( device, 0.1f, 0.1f, 0.1f );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    // plot an object at (0.9, 0.9, 0.9) that the user will look at
    error = tobii_calibration_collect_data_3d( device, 0.9f, 0.9f, 0.9f );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    // plot an object at (0.5, 0.5, 0.5) that the user will look at
    error = tobii_calibration_collect_data_3d( device, 0.5f, 0.5f, 0.5f );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    error = tobii_calibration_compute_and_apply( device );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    return tobii_calibration_stop( device );
}

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 = create_device( api, &device, url );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Failed to create device!\n" );
        tobii_api_destroy( api );
        return error;
    }

    error = calibrate( device );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "Calibration succeed!\n" );
    }
    else
    {
        printf( "Calibration failed!\n" );
    }

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

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 tobii_calibration_collect_data_per_eye_2d( tobii_device_t* device,
    float x, float y, tobii_enabled_eye_t requested_eyes,
    tobii_enabled_eye_t* collected_eyes );
```

Remarks This function should not be called unless the calibration process has been started by calling tobii_calibration_start().

This function is synchronous and may block for several seconds while the eye tracker collects calibration data.

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 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 which 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, tobii_calibration_collect_data_per_eye_2d returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_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_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The functionality is not supported on this device.

- **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_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_per_eye_2d from within a callback function is not supported.

- **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_clear(), tobii_calibration_collect_data_2d(), tobii_calibration_collect_data_3d(), 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_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example

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

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <assert.h>
#include <string.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!\n" );
        return 0;
    }

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

    if( file_size <= 0 )
    {
        fclose( license_file );
        printf( "Unable to read License file!\n" );
        return 0;
    }

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

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

tobii_error_t create_device( tobii_api_t* api, tobii_device_t** device, char* url )
{
    tobii_error_t 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 license_result;
    error = tobii_device_create_ex( api, url, TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE, &license, 1, &license_result,
device );
    free( license_key );
    license_key = 0;

    if( error == TOBII_ERROR_CONNECTION_FAILED )
    {
        printf( "Failed to connect to tracker.\n" );
        return error;
    }

    return error;
}

tobii_error_t calibrate( tobii_device_t* device, tobii_enabled_eye_t requested_eye )
{
    tobii_error_t error = tobii_calibration_start( device, TOBII_ENABLED_EYE_BOTH );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    printf( "Look at the top left!\n" );
    tobii_enabled_eye_t collected_eyes;
    error = tobii_calibration_collect_data_per_eye_2d( device, 0.1f, 0.1f, requested_eye,
        &collected_eyes );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    printf( "Look at the bottom right!\n" );
    error = tobii_calibration_collect_data_per_eye_2d( device, 0.9f, 0.9f, requested_eye,
        &collected_eyes );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    printf( "Look at the center!\n" );
    error = tobii_calibration_collect_data_per_eye_2d( device, 0.5f, 0.5f, requested_eye,
        &collected_eyes );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    error = tobii_calibration_compute_and_apply_per_eye( device, &collected_eyes );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    error = tobii_calibration_stop( device );
    if( error != TOBII_ERROR_NO_ERROR ) { return error; }

    if( collected_eyes != requested_eye ) { return TOBII_ERROR_OPERATION_FAILED; }

    return TOBII_ERROR_NO_ERROR;
}

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 = create_device( api, &device, url );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Failed to create device!\n" );
        tobii_api_destroy( api );
        return error;
    }

    error = calibrate( device, TOBII_ENABLED_EYE_LEFT );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "Left eye calibration succeed!\n" );
    }
    else
    {
        printf( "Left eye calibration failed!\n" );
    }

    error = calibrate( device, TOBII_ENABLED_EYE_RIGHT );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "Right eye calibration succeed!\n" );
    }
    else
    {
```

```
        printf( "Right eye calibration failed!\n" );
    }

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_calibration_discard_data_2d

Function

Discards all data collected for the specified screen coordinate.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_calibration_discard_data_2d( tobii_device_t* device,
    float x, float y );
```

Remarks

This function should not be called unless the calibration process has been started by calling tobii_calibration_start().

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 discard data for, as specified in a prior call to tobii_calibration_collect_data_2d.

y the y-coordinate (vertical) of the point to discard data for, as specified in a prior call to tobii_calibration_collect_data_2d.

Return value

If the operation is successful, tobii_calibration_discard_data_2d returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_calibration_discard_data_2d returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **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_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_2d from within a callback function is not supported.

- **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_clear(), tobii_calibration_collect_data_2d(), tobii_calibration_collect_data_3d(), tobii_calibration_collect_data_per_eye_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_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example

See tobii_calibration_collect_data_2d().

tobii_calibration_discard_data_3d

Function

Discards all data collected for the specified 3d coordinate.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_calibration_discard_data_3d( tobii_device_t* device,
    float x, float y, float z );
```

Remarks

This function should not be called unless the calibration process has been started by calling tobii_calibration_start().

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 discard data for, as specified in a prior call to tobii_calibration_collect_data_3d.

y the y-coordinate (vertical) of the point to discard data for, as specified in a prior call to tobii_calibration_collect_data_3d.

z the z-coordinate (depth) of the point to discard data for, as specified in a prior call to tobii_calibration_collect_data_3d.

Return value

If the operation is successful, tobii_calibration_discard_data_3d returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_calibration_discard_data_3d returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameter was passed in as NULL.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **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_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_3d from within a callback function is not supported.

- **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_clear(), 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_per_eye_2d(), tobii_calibration_compute_and_apply(), tobii_calibration_compute_and_apply_per_eye(), tobii_calibration_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example See tobii_calibration_collect_data_3d().

tobii_calibration_discard_data_per_eye_2d

Function Discards all data collected by a corresponding call to tobii_calibration_collect_data_per_eye_2d for a given stimuli point.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_calibration_discard_data_per_eye_2d( tobii_device_t* device,
    float x, float y, tobii_enabled_eye_t eyes );
```

Remarks This function should not be called unless the calibration process has been started by calling tobii_calibration_start().

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 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_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The functionality is not supported on this device.

- **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_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_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_clear(), 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_compute_and_apply(), tobii_calibration_compute_and_apply_per_eye(), tobii_calibration_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example See tobii_calibration_collect_data_per_eye_2d().

tobii_calibration_clear

Function Resets the calibration to the default state.

Syntax

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

Remarks Also performed internally when calling tobii_calibration_start.

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_INSUFFICIENT_LICENSE**

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

- **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_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_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_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example See tobii_calibration_collect_data_2d() and tobii_calibration_collect_data_3d().

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 This function should not be called unless the calibration process has been started by calling tobii_calibration_start().

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_INSUFFICIENT_LICENSE**

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

- **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_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_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_clear(), 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_per_eye(), tobii_calibration_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example See tobii_calibration_collect_data_2d() and tobii_calibration_collect_data_3d().

tobii_calibration_compute_and_apply_per_eye

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_per_eye( tobii_device_t* device,
    tobii_enabled_eye_t* calibrated_eyes );
```

Remarks This function should not be called unless the calibration process has been started by calling tobii_calibration_start().

Computes the calibration given the data gathered by calling tobii_calibration_discard_data_per_eye_2d(). Depending on the quality of the data gathered it will determine and return the information regarding for which eye the calibration was successful.

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_INSUFFICIENT_LICENSE**

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

- **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_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_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_clear(), 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_per_eye(), tobii_calibration_apply(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example

See tobii_calibration_collect_data_per_eye_2d().

tobii_calibration_retrieve

Function

Retrieves the currently applied calibration from the device as a binary blob.

Syntax

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

Remarks

This function will always return the binary data of the latest applied calibration, this data can be stored and then be used for re-applying the calibration in case of power cycle or switching user.

This function should be called after tobii_calibration_compute_and_apply()/tobii_calibration_compute_and_apply_per_eye() in order to retrieve the latest calibration data, otherwise the returned calibration will be the data from previous calibration.

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 data_receiver( void const* data, size_t size, void* user_data )
```

This function will be called with the retrieved calibration data. It is called with the following parameters:

- *data* 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.
- *size* The size of the calibration data, in bytes.
- *user_data* This is the custom pointer passed to tobii_calibration_retrieve.

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

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **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_retrieve from within a callback function is not supported.

- **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_apply(), tobii_calibration_parse()

Example

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

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <assert.h>
#include <string.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!\n" );
        return 0;
    }

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

```
    if( file_size <= 0 )
    {
        fclose( license_file );
        printf( "Unable to read License file!\n" );
        return 0;
    }

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

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

tobii_error_t create_device( tobii_api_t* api, tobii_device_t** device, char* url )
{
    tobii_error_t 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 license_result;
    error = tobii_device_create_ex( api, url, TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE, &license, 1, &license_result,
device );
    free( license_key );
    license_key = 0;

    if( error == TOBII_ERROR_CONNECTION_FAILED )
    {
        printf( "Failed to connect to tracker.\n" );
        return error;
    }

    return error;
}

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 );
}

typedef struct
{
    size_t size;
    uint8_t* data;
} calibration_blob_t;

void calibration_retrieve_callback( void const* data, size_t size, void* user_data )
{
    calibration_blob_t* blob = (calibration_blob_t*)user_data;
    blob->data = (uint8_t*)malloc( size * sizeof( blob->data ) );
    blob->size = size;
    memcpy(blob->data, data, size );
}

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 = create_device( api, &device, url );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Failed to create device!\n" );
        tobii_api_destroy( api );
        return error;
    }

    // Performed the calibration process and apply as the current calibration
    // with tobii_compute_and_apply();

    calibration_blob_t blob;
    error = tobii_calibration_retrieve( device, calibration_retrieve_callback, &blob );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        FILE* fp = fopen("calibration_blob.bin", "w+");
        fwrite( blob.data, sizeof(blob.data), blob.size, fp );
        fclose( fp );
        printf( "Calibration successfully stored on disk!\n" );
    }
    else
```

```
    {
        printf( "Calibration retrieval failed!\n" );
    }

    if( blob.data ) free( blob.data );

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_calibration_parse

Function

Extracts data about calibration points from the specified calibration.

Syntax

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

Remarks

Query which stimuli points where used in a specific calibration and gives information regarding if the stimuli point is valid and whether it was used in the calibration or not. Also contains information about the left and right eye focus point for each stimuli point.

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

data is the calibration data retrieved by tobii_calibration_retrieve().

data_size is the size of the data retrieved by tobii_calibration_retrieve().

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

```
void receiver( tobii_calibration_point_data_t const* point_data, void* user_data )
```

This function will be called for each parsed point from the calibration. It is called with the following parameters:

- *point_data* A pointer to a struct containing all the data related to a calibration point.
  - *point_xy*

    x and y coordinates of the requested calibration stimuli point.
  - *left_status*

    Indicates the left eye calibration status of the stimuli point:
    TOBII_CALIBRATION_POINT_STATUS_FAILED_OR_INVALID,
    TOBII_CALIBRATION_POINT_STATUS_VALID_BUT_NOT_USED_IN_CALIBRATION or
    TOBII_CALIBRATION_POINT_STATUS_VALID_AND_USED_IN_CALIBRATION
  - *left_mapping_xy*

    The left eye focus point captured during calibration.
  - *right_status*

    Indicates the right eye calibration status of the stimuli point:
    TOBII_CALIBRATION_POINT_STATUS_FAILED_OR_INVALID,
    TOBII_CALIBRATION_POINT_STATUS_VALID_BUT_NOT_USED_IN_CALIBRATION or
    TOBII_CALIBRATION_POINT_STATUS_VALID_AND_USED_IN_CALIBRATION
  - *right_mapping_xy*

    The right eye focus point captured during calibration.
- *user_data* This is the custom pointer sent in to tobii_calibration_parse.

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

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *api*, *data* or *receiver* parameters were passed in as NULL, or *data_size* parameter was less than 8.
- **TOBII_OPERATION_FAILED**

  The data being parsed was not a valid calibration.

See also

tobii_calibration_start(), tobii_calibration_stop(), tobii_calibration_clear(), 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_apply(), tobii_calibration_retrieve(),

Example

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

#include <stdio.h>
```

```
#include <stdlib.h>
#include <memory.h>
#include <assert.h>

void point_receiver( tobii_calibration_point_data_t const* point_data, void* user_data )
{
    (void)user_data;
    printf( "Target point : [%f, %f] ",
        point_data->point_xy[ 0 ], point_data->point_xy[ 1 ] );
    printf( "Left eye focus point : [%f, %f] ",
        point_data->left_mapping_xy[ 0 ], point_data->left_mapping_xy[ 1 ] );
    printf( "Right eye focus point : [%f, %f] \n",
        point_data->right_mapping_xy[ 0 ], point_data->right_mapping_xy[ 1 ] );
}

void calibration_retrieve_callback( void const* data, size_t size, void* user_data )
{
    tobii_api_t* api = (tobii_api_t*)user_data;

    tobii_error_t error = tobii_calibration_parse( api, data, size, point_receiver, NULL );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Parsing calibration failed!" );
        return;
    }
}

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_api_t* api = 0; // Dummy for demo purposes
    tobii_device_t* device = 0; // Dummy for demo purposes

    // Make the calibration and apply as the current calibration with tobii_compute_and_apply();

    tobii_error_t error = tobii_calibration_retrieve( device, calibration_retrieve_callback, api );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Retrieving calibration failed!" );
    }

    // destroy api and device, omitted here for brevity
    return error;
}
```

tobii_calibration_apply

Function Applies the specified calibration binary blob to the device, making it the current calibration.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_calibration_apply( tobii_device_t* device,
    void const* data, size_t size );
```

Remarks This function is used in conjunction with the tobii_calibration_retrieve() function to re-apply a previously retrieved binary data calibration blob.

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

data is the calibration data which has previously been retrieved by calling tobii_calibration_retrieve()

size is the size of the calibration data which has previously been retrieved by calling tobii_calibration_retrieve()

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *data* parameters were passed in as NULL, or the *data_size* parameter was not greater than 0.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

  The provided calibration could not be applied to the device.

- **TOBII_ERROR_CALIBRATION_BUSY**

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

- **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_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_clear(), tobii_calibration_retrieve(), tobii_calibration_parse()

Example

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

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <assert.h>
#include <string.h>

static size_t read_file( uint16_t* buffer, char* file_path )
{
    FILE *fp = fopen( file_path, "rb" );

    if( !fp )
    {
        printf( "File could not be found!\n" );
        return 0;
    }

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

    if( file_size <= 0 )
    {
        fclose( fp );
        printf( "Unable to read file!\n" );
        return 0;
    }

    if( buffer )
    {
        fread( buffer, sizeof( uint16_t ), file_size / sizeof( uint16_t ), fp );
    }

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

tobii_error_t create_device( tobii_api_t* api, tobii_device_t** device, char* url )
{
    tobii_error_t error;

    size_t license_size = read_file( 0, "se_license_key_sample");
    assert( license_size > 0 );

    uint16_t* license_key = (uint16_t*)malloc( license_size );
    memset( license_key, 0, license_size );
    read_file( license_key, "se_license_key_sample" );
    tobii_license_key_t license = { license_key, license_size };

    tobii_license_validation_result_t license_result;
    error = tobii_device_create_ex( api, url, TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE, &license, 1, &license_result,
device );
    free( license_key );
    license_key = 0;

    if( error == TOBII_ERROR_CONNECTION_FAILED )
    {
        printf( "Failed to connect to tracker.\n" );
        return error;
    }

    return error;
}

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 = create_device( api, &device, url );
    if( error != TOBII_ERROR_NO_ERROR )
```

```
    {
        printf( "Failed to create device!\n" );
        tobii_api_destroy( api );
        return error;
    }

    // See example for tobii_calibration_retrieve() on how to fetch and store the binary calibration blob

    size_t calibration_size = read_file( 0, "calibration_blob.bin" );
    assert( calibration_size > 0 );
    uint16_t* calibration_data = (uint16_t*)malloc( calibration_size );
    memset( calibration_data, 0, calibration_size );
    read_file( calibration_data, "calibration_blob.bin" );

    error = tobii_calibration_apply( device, calibration_data, calibration_size );
    free( calibration_data );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "Calibration was successfully applied!\n" );
    }
    else
    {
        printf( "Calibration apply failed!\n" );
    }

    tobii_device_destroy( device );
    tobii_api_destroy( api );

    return 0;
}
```

tobii_get_geometry_mounting

Function

Queries the mounting geometry of a remote eye tracker

Syntax

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

Remarks

This information is needed when calculating the display area screen plane coordinates.

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. The tobii_geometry_mounting_t contains the following values:

- *guides*

  Integer value containing the number of display alignment guide lines on a remote trackers. These guides should be duplicated in the display setup UI so they can be aligned with the physical guide grooves on the tracker.

- *width_mm*

  Floating point value containing the width of a remote eye tracker in mm. If *guides* > 1, then the value contains the distance between two guides.

- *angle_deg*

  Floating point value containing the tilt angle of a remote tracker in degrees.

- *external_offset_mm_xyz*

  Floating point array containing the 3d-coordinate of the offset between the bottom center of the screen and the point at the top front (x = 0) of the eye tracker enclosure

- *internal_offset_mm_xyz*

  Floating point array containing the 3d-coordinates of the offset along the y- and z-axis from the eye tracker center point (origin in the eye tracker coordinate system) to a point at the top front of the eye tracker enclosure, at the intersection of the centerline (x = 0).

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_INTERNAL**

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

See also

tobii_calculate_display_area_basic()

Example

See tobii_set_display_area().

tobii_get_display_area

Function

Retrieves the current display area coordinates from the device.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_get_display_area( tobii_device_t* device,
    tobii_display_area_t* display_area );
```

Remarks

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

display_area must be a valid pointer to a tobii_display_area_t instance which will receive the result. tobii_display_area_t contains the following values:

- *top_left_mm_xyz*

  Floating point array containing the 3d-coordinate of the top left screen plane.

- *top_right_mm_xyz*

  Floating point array containing the 3d-coordinate of the top right screen plane.

- *bottom_left_mm_xyz*

  Floating point array containing the 3d-coordinate of the bottom left screen plane.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **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_area from within a callback function is not supported.

- **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_display_area(), tobii_calculate_display_area_basic()

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()
    // and on how to create/destroy a device using tobii_device_create/destroy().
    tobii_device_t* device = 0; // Dummy for demo purposes

    tobii_display_area_t display_area;
    tobii_error_t error = tobii_get_display_area( device, &display_area );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Failed to retrieve display area!\n" );
        // destroy api and device, omitted here for brevity
        return error;
    }

    printf( "Currently configured display area:\n" );
    printf( "bottom left = [%f, %f, %f]\n",
        display_area.bottom_left_mm_xyz[0], display_area.bottom_left_mm_xyz[1], display_area.bottom_left_mm_xyz[2] );
    printf( "top left = [%f, %f, %f]\n",
        display_area.top_left_mm_xyz[0], display_area.top_left_mm_xyz[1], display_area.top_left_mm_xyz[2] );
    printf( "top right = [%f, %f, %f]\n",
        display_area.top_right_mm_xyz[0], display_area.top_right_mm_xyz[1], display_area.top_right_mm_xyz[2] );
```

```
    // destroy api and device, omitted here for brevity
    return 0;
}
```

tobii_set_display_area

Function

Configures the display area of the device.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_set_display_area( tobii_device_t* device,
    tobii_display_area_t const* display_area );
```

Remarks

Configures the display area of the current eye tracker/monitor setup. It is defined by specifying the three corner coordinates that makes up the screen plane, top left, lower left and top right. The helper function tobii_calculate_display_area_basic() provides a convenient way of calculating the display area for basic eye tracker/monitor setups.

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

display_area must be a valid pointer to a tobii_display_area_t which is correctly initialize, for example by calling tobii_calculate_display_area_basic(). tobii_display_area_t contains the following values:

- *top_left_mm_xyz*

  Floating point array containing the 3d-coordinate of the top left screen plane.

- *top_right_mm_xyz*

  Floating point array containing the 3d-coordinate of the top right screen plane.

- *bottom_left_mm_xyz*

  Floating point array containing the 3d-coordinate of the bottom left screen plane.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **TOBII_ERROR_CALIBRATION_BUSY**

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

- **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_set_display_area from within a callback function is not supported.

- **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_display_area(), tobii_calculate_display_area_basic()

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_api_t* api = 0; // Dummy for demo purposes
    tobii_device_t* device = 0; // Dummy for demo purposes

    tobii_geometry_mounting_t geometry_mounting;
    tobii_error_t error = tobii_get_geometry_mounting( device, &geometry_mounting );
    if( error != TOBII_ERROR_NO_ERROR )
```

```
    {
        printf( "Failed to retrieve geometry mounting!\n" );
        // destroy api and device, omitted here for brevity
        return error;
    }

    // Screen geometry, usually obtained via a UI application
    float width_mm = 300.0f;
    float height_mm = 200.0f;
    float offset_x_mm = 10.0f;

    tobii_display_area_t display_area;
    error = tobii_calculate_display_area_basic( api, width_mm, height_mm, offset_x_mm, &geometry_mounting, &display_area
);
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Failed to calculate display area!\n" );
        // destroy api and device, omitted here for brevity
        return error;
    }

    error = tobii_set_display_area( device, &display_area );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        printf( "Failed to calculate display area!\n" );
        // destroy api and device, omitted here for brevity
        return error;
    }

    printf( "Display area successfully configured!\n" );

    // destroy api and device, omitted here for brevity
    return 0;
}
```

tobii_calculate_display_area_basic

Function

Calculates a basic display area configuration based on screen size and geometry mounting.

Syntax

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

Remarks

Calculate the display area for a basic eye tracker/monitor setup, where the eye tracker is situated just beneath the screen plane, only offset from center screen along the x-axis. The resulting tobii_display_area_t struct can be used as an input parameter to tobii_set_display_area().

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

width_mm is the width of the display device in millimeters.

height_mm is the height of the display device in millimeters.

offset_x is the offset of the eye tracker from the center of the display device in the x-axis, in millimeters.

geometry_mounting is the geometry mounting information as received by tobii_get_geometry_mounting()

display_area must be a valid pointer to a tobii_display_area_t instance which will receive the result.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *api*, *geometry_mounting* or *display_area* parameters was passed in as NULL.

See also

tobii_set_display_area(), tobii_get_geometry_mounting()

Example

See tobii_set_display_area()

tobii_get_device_name

Function

Retrieves the custom name for a given eye tracker device.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_get_device_name( tobii_device_t* device,
    tobii_device_name_t* device_name );
```

Remarks

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

device_name must be a valid pointer to a tobii_device_name_t instance which will receive the result.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **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_device_name from within a callback function is not supported.

- **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_device_name()

Example

```
#include <tobii/tobii_config.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_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_device_name_t device_name;
    error = tobii_get_device_name( device, &device_name );
    if( error == TOBII_ERROR_NOT_SUPPORTED )
    {
        printf( "Get device name is not supported on this tracker \n" );
    }
    else
    {
        assert( error == TOBII_ERROR_NO_ERROR );
        printf( "Name of the device is: %s \n", device_name );
    }

    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_device_name

Function Configures a custom name for a given eye tracker device.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_set_device_name( tobii_device_t* device,
    tobii_device_name_t const device_name );
```

Remarks Configures a custom name for a given eye tracker device, to help distinguish between different eye trackers, i.e. in a lab environment with multiple connected eye trackers.

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

device_name a tobii_device_name_t containing the name to be applied.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **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_set_device_name from within a callback function is not supported.

- **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_device_name()

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; // Dummy for demo purposes

    tobii_device_name_t name = "Tobii Tracker";
    tobii_error_t error = tobii_set_device_name( device, name );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "Device name has been set!\n" );
    }
    else
    {
        printf( "Device name cold not be set!\n" );
    }

    // destroy api and device, omitted here for brevity
    return 0;
}
```

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

List all output frequencies supported by the advanced gaze data stream, which can then be used as input to tobii_set_output_frequency().

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_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **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_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_output_frequency(), tobii_get_output_frequency()

Example

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

typedef struct output_frequency_context_t
{
    float output_frequencies[ 10 ];
    size_t count;
} output_frequency_context_t;

void output_frequency_receiver( float output_frequency, void* user_data )
{
    if ( user_data == NULL ) return;

    output_frequency_context_t* output_frequency_context = (output_frequency_context_t*)user_data;
    output_frequency_context->output_frequencies[ output_frequency_context->count++ ] = output_frequency;
}

int main()
{
    // See tobii.h for examples on how to create/destroy an api using tobii_api_create/destroy()
    // and on how to create/destroy a device using tobii_device_create/destroy().
    tobii_device_t* device = 0; // Dummy for demo purposes

    output_frequency_context_t output_frequency_context = {0};
    tobii_error_t error = tobii_enumerate_output_frequencies( device, output_frequency_receiver,
        &output_frequency_context );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "* This tracker supports %zu output frequencies: ", output_frequency_context.count );

        for( size_t i = 0; i < output_frequency_context.count; i++ )
        {
            printf( "%f%c " , output_frequency_context.output_frequencies [ i ],
                i + 1 < output_frequency_context.count ? ',' : '.' );
        }
    }
    else
    {
        printf( "* Failed to get supported output frequencies for this tracker.\n" );
    }

    // destroy api and device, omitted here for brevity
    return 0;
}
```

tobii_set_output_frequency

Function

Configures the output frequency of the advanced gaze data stream.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_set_output_frequency( tobii_device_t* device,
    float output_frequency );
```

Remarks

Configures the output frequency of the advanced gaze data stream, i.e. how many gaze data samples will be sent to the client per second, the supported output frequencies can be listed by calling tobii_enumerate_output_frequency().

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

output_frequency the frequency to apply.

Return value

If the operation is successful, tobii_set_output_frequency returns **TOBII_ERROR_NO_ERROR**. If the call fails,

tobii_set_output_frequency returns one of the following:

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* parameters were passed in as NULL, or *output_frequency* is not valid. To list all valid output frequencies for the device, use the tobii_enumerate_output_frequencies() function.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

  The function failed because of the one of following reasons:

  - the function was called while the device was in calibration mode;
  - the function was called while some user of the device was subscribed to a stream that does not support the new output frequency.

- **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_set_output_frequency from within a callback function is not supported.

- **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_output_frequency(), tobii_enumerate_output_frequencies()

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; // Dummy for demo purposes

    float output_frequency = 33.0f;
    tobii_error_t error = tobii_set_output_frequency( device, output_frequency );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "Output frequency set!\n" );
    }
    else
    {
        printf( "Setting output frequency failed\n" );
    }

    // destroy api and device, omitted here for brevity
    return 0;
}
```

tobii_get_output_frequency

Function Query the current output frequency of the advanced gaze data stream.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_get_output_frequency( tobii_device_t* device,
    float* output_frequency );
```

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

output_frequency is a valid pointer to a float which will receive the current output frequency.

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **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_output_frequency from within a callback function is not supported.

- **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_output_frequency(), tobii_enumerate_output_frequencies()

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()
    // and on how to create/destroy a device using tobii_device_create/destroy().
    tobii_device_t* device = 0; // Dummy for demo purposes

    float output_frequency = 0.0f;
    tobii_error_t error = tobii_get_output_frequency( device, &output_frequency );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "* Current output frequency is %f Hz\n", output_frequency );
    }
    else
    {
        printf( "Get output frequency failed\n" );
    }

    // destroy api and device, omitted here for brevity
    return 0;
}
```

tobii_output_frequency_subscribe

Function Start listening for changes to the output frequency property.

Syntax

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

Remarks This subscription is for receiving notifications on changes to the output frequency property.

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 tobii_output_frequency_receiver( const float output_frequency, void* user_data )
```

This function will be called when the output frequency property has changed. It is called with the following parameters:

- *output_frequency*

  This is float data containing the output frequency.

- *user_data* This is a pointer to custom user data sent in when registering the callback in the call to tobii_output_frequency_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_output_frequency_subscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_output_frequency_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_output_frequency_receiver_t function.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

The functionality is not supported on this device.

- **TOBII_ERROR_ALREADY_SUBSCRIBED**

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

- **TOBII_ERROR_OPERATION_FAILED**

  The output frequency property change notification subscribe operation failed.

- **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_output_frequency_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>
#include <string.h>

void output_frequency_callback( float output_frequency, void* user_data )
{
    (void)user_data;
    printf( "Output frequency: %f\n", output_frequency );
}

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

    error = tobii_output_frequency_subscribe( device, output_frequency_callback, NULL );
    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_output_frequency_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_output_frequency_unsubscribe

Function Stops listening to notifications on changes to output frequency property that was subscribed to by a call to tobii_output_frequency_subscribe()

Syntax

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

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

Return value If the operation is successful, tobii_output_frequency_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails,

tobii_output_frequency_unsubscribe returns an error code specific to the device.

See also tobii_output_frequency_subscribe()

Example See tobii_output_frequency_subscribe()

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_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **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_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_display_id()

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()
    // and on how to create/destroy a device using tobii_device_create/destroy().
    tobii_device_t* device = 0; // Dummy for demo purposes

    tobii_display_id_t display_id;
    tobii_error_t error = tobii_get_display_id( device, &display_id );
    if( error != TOBII_ERROR_NO_ERROR )
    {
        // destroy api and device, omitted here for brevity
        printf( "Retrieving display id failed!\n" );
        return error;
    }
    printf( "Display ID = %s\n", display_id );
    // destroy api and device, omitted here for brevity
    return 0;
}
```

tobii_set_display_id

Function Applies the specified display area setting to the device.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_set_display_id( tobii_device_t* device,
    tobii_display_id_t const* 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 a valid tobii_display_id_t containing the display id to be applied. The maximum size of the device id is 256 characters and it shall be '\0'-terminated.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The function failed because the operation is not supported by the connected tracker.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

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

- **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_set_display_id from within a callback function is not supported.

- **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_display_id()

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; // Dummy for demo purposes

    tobii_display_id_t display_id = "DUMMY_ID_123";
    tobii_error_t error = tobii_set_display_id( device, display_id );
    if( error == TOBII_ERROR_NO_ERROR )
    {
        printf( "Display ID has been set!\n" );
    }
    else
    {
        printf( "Display ID cold not be set!\n" );
    }

    // destroy api and device, omitted here for brevity
    return 0;
}
```

tobii_display_id_subscribe

Function

Start listening for changes to the display ID property.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_display_id_subscribe( tobii_device_t* device,
    tobii_display_id_callback_t callback, void* user_data );
```

Remarks

This subscription is for receiving notifications on changes to the display ID property.

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 tobii_display_id_callback( tobii_display_id_t const* display_id, void* user_data )
```

This function will be called when the display id property has changed. It is called with the following parameters:

- *display_id*

  This is a pointer to a char array of size 256 containing the '\0'-terminated device id.

- *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 blocklisted.

- **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_OPERATION_FAILED**

  The display id property change notification subscribe operation failed.

- **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_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>
#include <string.h>

void display_id_callback( tobii_display_id_t const display_id, void* user_data )
{
    (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_STORE_OR_TRANSFER_FALSE, &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

Function

Stops listening to notifications on changes to display id property that was subscribed to by a call to tobii_display_id_subscribe()

Syntax

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

Remarks

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

Return value

If the operation is successful, tobii_display_id_unsubscribe returns **TOBII_ERROR_NO_ERROR**. If the call fails, tobii_display_id_unsubscribe returns an error code specific to the device.

See also

tobii_display_id_subscribe()

Example

See tobii_display_id_subscribe()

tobii_calibration_stimulus_points_get

Function

Retrieves information about the stimulus points used to compute the personal calibration currently in use.

Syntax

```
#include <tobii/tobii_config.h>
tobii_error_t tobii_calibration_stimulus_points_get( tobii_device_t* device,
    tobii_calibration_stimulus_points_t* stimulus_points  );
```

Remarks

A personal calibration is based on gaze data collected for a number of fixed stimuli points with a known position on the display plane (remote eye trackers) or in space (wearable eye trackers). This function can be used to retrieve information about the personal calibration currently applied on the eye tracker. The information includes information about what stimuli points that were used when doing the personal calibration, as well both an offset in each point and the precision of the gaze data in each point. The values can be used as an indication how well the eye tracker was able to track the user in the area around the stimuli point position.

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

stimulus_points is a valid pointer to a tobii_calibration_stimulus_points_t instance to receive the information. It constains the following fields:

- *stimulus_point_count* is the number of tobii_calibration_stimulus_point_data_t.
- *stimulus_points* one array of tobii_calibration_stimulus_point_data_t, which contains the following information:
  - *point_xyz* It is the calibration stimulus point coordinate specified during calibration, for wearable trackers this coordinate relates to a point in 3d space, for remote trackers the coordinate relates to a point where the x and y coordinates are on the screen plane, where z-axis is the offset from the screen plane in the direction of the normal vector.
  - *left_status/right_status* The status of this stimuli point, includes information whether this stimulus point was used for setting the personal calibration and the type of the problem that may be the cause for it not to be used. It is one of tobii_calibration_stimulus_point_status_t enum value:

```
**TOBII_CALIBRATION_STIMULUS_POINT_STATUS_FAILED_OR_INVALID**
  No gaze data could be calculated for this stimulus point.

**TOBII_CALIBRATION_STIMULUS_POINT_STATUS_VALID_NOT_USED**
  Gaze data could be calculated for this stimulus point, but the data in this point was not used for
personal calibration. This can happen for example if the user is
  not fixating at the point or if the user is looking at some point of the screen.

**TOBII_CALIBRATION_STIMULUS_POINT_STATUS_VALID_USED*
  This stimuli point is used in the calibration.
```

  - *left_bias/right_bias* The distance between stimulus point and mean of the gaze point that was collected during personal calibration. For a remote eye tracker this will be a value in mm in the display plane, for wearable trackers it will be an angle in degrees.
  - *left_precision/right_precision* The standard deviation of the difference between the gaze vectors and the mean gaze. For remote trackers this the distance in mm in the display will be used and for wearable trackers, angular differences will be used to calculate the standard deviation.

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  Getting calibration stimulus points property is not supported by the device.

- **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_stimulus_points_get from within a callback function is not supported.

- **TOBII_ERROR_INTERNAL**

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

Example

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

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

    tobii_calibration_stimulus_points_t stimulus_points;

    error = tobii_calibration_stimulus_points_get( device, &stimulus_points );
    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_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

Function Starts the gaze data stream.

Syntax

```
#include <tobii/tobii_advanced.h>
tobii_error_t tobii_gaze_data_subscribe( tobii_device_t* device,
    tobii_gaze_data_callback_t callback, void* user_data );
```

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.

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

Function Stops the gaze data stream.

Syntax

```
#include <tobii/tobii_advanced.h>
tobii_gaze_data_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 fails, tobii_gaze_data_unsubscribe returns an error code specific to the device.

See Also tobii_gaze_data_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_gaze_data_callback(  tobii_gaze_data_t const* gaze_data, void* user_data  )
{
    (void)user_data;
    if( gaze_data->right.gaze_point_validity == TOBII_VALIDITY_VALID )
        printf( "Gaze point (right): %f, %f\n",
        gaze_data->right.gaze_point_on_display_normalized_xy[ 0 ],
        gaze_data->right.gaze_point_on_display_normalized_xy[ 1 ] );
    else
        printf( "Gaze point (right): INVALID\n");

    if( gaze_data->left.gaze_point_validity == TOBII_VALIDITY_VALID )
        printf( "Gaze point (left): %f, %f\n",
        gaze_data->left.gaze_point_on_display_normalized_xy[ 0 ],
        gaze_data->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( &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_STORE_OR_TRANSFER_FALSE, &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

Function

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.

Syntax

```
#include <tobii/tobii_advanced.h>
tobii_error_t tobii_digital_syncport_subscribe( tobii_device_t* device,
    tobii_digital_syncport_callback_t callback, void* user_data );
```

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.

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

```
void digital_syncport_callback( uint32_t signal, int64_t timestamp_tracker_us,
    int64_t timestamp_system_us, void* user_data )

This function will be called when the syncport data value changes. It is called with the following parameters:

-   *signal*
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.

-   *timestamp_tracker_us*
Timestamp value for when the digital syncport 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 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
tobii_system_clock can be used to retrieve a timestamp using the same clock and same relative values as this
timestamp.

-   *user_data* 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 has failed one of the following errors 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 blocklisted.

- **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_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.

- **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.

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 errors 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 blocklisted.

- **TOBII_ERROR_NOT_SUBSCRIBED**

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

- **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.

- **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_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 tacker
}

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_STORE_OR_TRANSFER_FALSE, &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_timesync

Function

Establishes a synchronization point between two clocks: eye tracker clock and system host clock. Based on the *timesync* data, we can compute the offset between the two clocks (see the example bellow).

Syntax

```
#include <tobii/tobii_advanced.h>
tobii_timesync( tobii_device_t* device, tobii_timesync_data_t* timesync );
```

Remarks

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

timesync is a valid pointer to a tobii_timesync_data_t instance to receive the information.

It constains the following fields:

- *system_start_us*: initial timestamp from the host system clock.
- *system_end_us*: second timestamp from the host system clock.
- *tracker_us*: timestamp from the eye tracker clock.

system_start_us and *system_end_us* are used to calculate the round trip time (RTT) for the time sync operation. The synchronization point is where *tracker_us* represents the same time as *system_start_us* + RTT / 2

Return value

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

- **TOBII_ERROR_INVALID_PARAMETER**

  The *device* or *timesync* parameters have been passed in as NULL.

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUBSCRIBED**

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

- **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_timesync from within a callback function is not supported.

- **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**

  The functionality is not supported on this device.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_CONNECTION_FAILED_DRIVER**

  A connection failure occurred in the platform runtime driver.

- **TOBII_ERROR_FIRMWARE_UPGRADE_IN_PROGRESS**

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

Example

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

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

// When timesync packages have higher round trip time than this they are not usable
#define TIMESYNC_MAX_ROUNDTRIP_TIME_US 6000

int main()
{
    tobii_api_t* api;
    int64_t quality_us, tracker_offset_us;

    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 );

    // if not using a pro tracker use tobii_device_create_ex with Professional license
    error = tobii_device_create( api, url, TOBII_FIELD_OF_USE_STORE_OR_TRANSFER_FALSE, &device );
    assert( error == TOBII_ERROR_NO_ERROR );

    tobii_timesync_data_t data = { 0 };

    error = tobii_timesync( device, &data );
    assert( error == TOBII_ERROR_NO_ERROR );

    quality_us = data.system_end_us - data.system_start_us; // round trip time (RTT) for the packet
    if( quality_us > TIMESYNC_MAX_ROUNDTRIP_TIME_US )
    {
        return 1; // if it is too big, it is not reliable
    }

    tracker_offset_us = ( data.system_end_us + data.system_start_us ) / 2 - data.tracker_us;
    (void) tracker_offset_us; // Unused in sample code

    tobii_device_destroy( device );
    tobii_api_destroy( api );
```

```
    return 0;
}
```

tobii_enumerate_face_types

Function Retreives all supported face types from a specific eye tracker.

Syntax

```
#include <tobii/tobii_advanced.h>
tobii_error_t tobii_enumerate_face_types( tobii_device_t* device, tobii_face_type_receiver_t receiver,
  void* user_data );
```

Remarks 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 were auto detecting such differences is difficult or dangerous.

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 face_type_receiver(const tobii_face_type_t face_type, void* user_data);

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

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

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

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The eye tracker does not support enumeration of face types.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **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_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

Function Applies the specified face type setting to the device.

Syntax

```
#include <tobii/tobii_advanced.h>
tobii_error_t tobii_set_face_type( tobii_device_t* device, tobii_face_type_t const face_type );
```

Remarks 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 *TOBII_NOTIFICATION_TYPE_FACE_TYPE_CHANGED* will be broadcasted to all clients notifying them that the face type has changed and that a new calibration has to be set.

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

face_type 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 tobii_enumerate_face_types() function.

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The device firmware has no support for setting face type.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **TOBII_ERROR_OPERATION_FAILED**

  The function failed because it was called while the device was in calibration mode.

- **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_set_face_type from within a callback function is not supported.

- **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_enumerate_face_types()

tobii_get_face_type

Function Retreives the current face type setting of the device.

Syntax

```
#include <tobii/tobii_advanced.h>
tobii_error_t tobii_get_face_type( tobii_device_t* device, tobii_face_type_t* face_type );
```

Remarks 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 were auto detecting such differences is difficult or dangerous.

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

face_type is a pointer to a zero-terminated string representation of the current face type setting, with a maximum length of 63 characters.

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

- **TOBII_ERROR_INVALID_PARAMETER**

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

- **TOBII_ERROR_INSUFFICIENT_LICENSE**

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

- **TOBII_ERROR_NOT_SUPPORTED**

  The device firmware has no support for retreiving the current face type.

- **TOBII_ERROR_CONNECTION_FAILED**

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

- **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_face_type from within a callback function is not supported.

- **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_face_type() and tobii_enumerate_face_types()

TobiiBinder.h

Android applications initiate the communication with the eyetracker through its java binder. For this reason, the Android applications must link against the java binder of the Tobii Stream Engine library (tobii_stream_engine.aar), where TobiiBinder is the public java interface. TobiiBinder is initialized through TobiiBinderFactory.create(). It contains functions to start/stop and to get the status of the Tobii runtime service for Android.

The native Android library part of the application must similarly link against the Tobii Stream Engine library (libtobii_stream_engine.so), load it, and then map the required API functions for the applications usage. See the separate example "Native Android Usage" at the end this document.

If you are using a custom service for Android you do not need to set up a binder connection.

Folder Structure

apk Folders/Files
app
src
main
cpp
CMakeLists.txt
jni.cpp
tobii.h
java
com
example
secaar
MainActivity.java
libs
tobii_stream_engine.aar
res
AndroidManifest.xml

How to Start/Stop the Runtime

start

Function

Starts the Tobii runtime service if it's not already started.

Syntax

```
import com.tobii.binder.*;
...
private TobiiBinder tb = TobiiBinderFactory.create();
...
tb.start(getApplicationContext(), cbImpl);
```

Remarks

This is an async call. When the Tobii runtime is ready after initialization (and firmware upgrade if needed) there will be a notification via the TobiiBinderCallbackHandler.ready interface function (implemented by cbImpl).

Return value

Returns True if the start call was successful. It is required to wait until the async call completes before calling any other stream engine API function. Returns False if there are errors or if the Tobii runtime service is not installed. Errors in starting the service will be returned via the TobiiBinderCallbackHandler.ready interface function's argument (implemented by cbImpl).

Example

```
//------------------------------------------------
// File: MainActivity.java
//------------------------------------------------

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import com.tobii.binder.*;


public class MainActivity extends AppCompatActivity implements TobiiBinderCallbackHandler
{

    private TobiiBinder tb = TobiiBinderFactory.create();

    //Override
    public void ready(TobiiBinderCallbackHandler.Error e)
    {
        Log.d(TAG, "runtime error:" + e.toString());
        startEt();
    }
    //Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
```

```
        tb.start(getApplicationContext(), this);
        setContentView(R.layout.activity_main);
    }
    //Override
    protected void onDestroy()
    {
        super.onDestroy();
        tb.stop(getApplicationContext());
    }

    static native void startEt();
    static
    {
        System.loadLibrary("native-lib");
    }
}
```

stop

Function Stops the Tobii runtime service if there are no other clients connected.

Syntax

```
import com.tobii.binder.*;
...
private TobiiBinder tb = TobiiBinderFactory.create();
...
tb.start(getApplicationContext(), cbImpl);
...
tb.stop(getApplicationContext());
```

Remarks This is an async call.

Return value Returns True if the stop call was successful. Returns False if there are errors.

See also public boolean start(Context ctx, TobiiBinderCallbackHandler cb)

Example See public boolean start(Context ctx, TobiiBinderCallbackHandler cb)

isReady

Function Indicates if the Tobii runtime service is ready. This is an alternative to callbacks. For example, C# code cannot implement the Java interface, so the callback parameter is set to null in the start function and no further API calls should be made until the isReady function returns true.

Syntax

```
import com.tobii.binder.*;
...
private TobiiBinder tb = TobiiBinderFactory.create();;
...
tb.start(getApplicationContext(), null);
...
while(!tb.isReady())
{
    Thread.sleep(1);
}
...
startEt();
...
tb.stop(getApplicationContext());
```

Remarks It is required to wait for the Tobii runtime to be ready before using other Tobii API functions. (If a custom service is used instead of the Tobii runtime service then the isReady function should not be called.)

Return value Returns True if the Tobii runtime service has started, otherwise returns False.

See also public boolean start(Context ctx, TobiiBinderCallbackHandler cb)

Example See public boolean start(Context ctx, TobiiBinderCallbackHandler cb)

Native Android Usage

Example

```
//-------------------------------------------------
//File: jni.cpp
//-------------------------------------------------
// JNI - Java Native Interface, interface between Java application and Native Application.

extern "C" JNIEXPORT jstring JNICALL Java_com_tobii_secsample_Sample_connect(JNIEnv* env, jobject obj,
                                                                             jstring file_path)
{
        // get the path to where android extracts native libraries to
        temp = ...
        std::string libpath(temp);
        init(libpath)
}

//-------------------------------------------------
//File: test.hpp
```

```
//------------------------------------------------
#include <tobii/tobii.h>
#include <android/log.h>

#define INIT_ERROR -1
#define INIT_SUCCESS 0

#define LOG_TAG "AppNative"
#define LOGD(...) __android_log_print(ANDROID_LOG_DEBUG, LOG_TAG, __VA_ARGS__)
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR, LOG_TAG, __VA_ARGS__)

class tests
{
    public:
    int init(std::string libPath);
        int create_api(...);
    private:
    void* lib_handle;
        tobii_api_t* api;

        tobii_error_t (*tobii_api_create)(tobii_api_t** api, tobii_custom_alloc_t const* custom_alloc,
                                    tobii_custom_log_t const* custom_log);
};

//------------------------------------------------
//File: test.cpp
//------------------------------------------------

#include <test.hpp>

static void do_log(void* log_context, tobii_log_level_t level, char const* text)
{
    (void)log_context;
    (void)level;

    LOGD("SEC %s\n", text);
}
int tests::init(std::string libPath)
{
    string flPath = libPath + "/libtobii_stream_engine.so";
    lib_handle = dlopen(flPath.c_str(), RTLD_LAZY);
    if (!lib_handle)
    {
        LOGE("Stream engine library opens failed \n");
        return INIT_ERROR;
    }

    tobii_api_create = (tobii_error_t(*)(tobii_api_t * *api, tobii_custom_alloc_t const* custom_alloc,
                                     tobii_custom_log_t const* custom_log)) dlsym(lib_handle, "tobii_api_create");
    .....
        return INIT_SUCCESS;
}

int tests::create_api(....)
{
    api = 0;
    custom_log.log_func = do_log;

        tobii_error_t error = tobii_api_create(&api, nullptr, &custom_log);
    .....
    return 0;
}



//------------------------------------------------
// File: CMakeLists.txt
//------------------------------------------------
//
// Native file can be linked using Cmake file as below

cmake_minimum_required(VERSION 3.10.2)

# Declares and names the project.

project("secaar1")

add_library( # Sets the name of the library.
            native-lib

            # Sets the library as a shared library.
            SHARED

            # Provides a relative path to your source file(s).
            jni.cpp)

# libtobii_stream_engine.so should be taken from .aar file, put under
# ${CMAKE_SOURCE_DIR}/../libs/${CMAKE_ANDROID_ARCH_ABI}/ and linked as shown below:
add_library( tobii_stream_engine-lib
SHARED IMPORTED GLOBAL) set_property(TARGET tobii_stream_engine-lib PROPERTY IMPORTED_LOCATION
"${CMAKE_SOURCE_DIR}/../libs/${CMAKE_ANDROID_ARCH_ABI}/libtobii_stream_engine.so")

target_link_libraries( # Specifies the target library.
                    native-lib

                    tobii_stream_engine-lib
                    # Links the target library to the log library
                    # included in the NDK.
```

log)