

A large, vertical, close-up image of a human eye, rendered in a warm, orange-yellow color scheme. The eye is looking slightly to the left, and the eyelashes are visible. The image has a textured, almost painterly quality.

Tobii SDK 3.0

Developers Guide

Release Candidate 1
June 14, 2011

This is the developers guide for the Tobii SDK. It contains the documentation necessary to start developing eyetracking software against Tobii eyetracking hardware.

Table of Contents

Table of Figures and Tables	7
Introduction.....	8
New and Noteworthy	8
High Level Changes	8
What is included in this release?	9
Eye tracker Hardware Support	9
Supported Eye trackers.....	9
Firmware Requirements	9
Release Deliverables	9
Windows	9
Ubuntu 10.04	9
Mac OS X.....	10
SDK Components	10
Getting Started	10
Getting Started on Windows	10
Before you unpack the SDK	10
Running the .NET Examples	11
Setting up the Python Environment	13
Getting Started on Ubuntu Linux 10.04 LTS	14
Prerequisites on Ubuntu 10.04 LTS	14
Running the examples on Ubuntu Linux 10.04 LTS	15
Getting Started on Mac OS X	17
Installing	17
Referencing TobiiSDK.....	17
Using TobiiSDK from mono	18
Developing SDK Applications.....	19
Library Initialization	19
Thread Handling in the Different Languages	19
C#/.Net Specifics.....	19
Python Specifics.....	19
C++ Specifics	20
Objective-C Specifics.....	21
Handling Errors and Exception	21
The EyetrackerException class.....	21
Cocoa error-handling.....	22
Some Notes on Coordinate Systems	22

Browsing for available eye trackers	23
Finding Eye Trackers	23
Eye tracker Properties	23
The EyetrackerBrowser class	24
C# Interface	24
Python Interface	25
C++ Interface	25
Objective-C interface	26
The EyetrackerInfo class	26
C# Interface	26
Python Interface	27
C++ Interface	28
Objective-C interface	28
Connecting to an Eye Tracker	29
C# Interface	29
Python Interface	30
C++ Interface	30
Objective-C Interface	30
Handling Connection Errors	31
C# Interface	31
Python Interface	31
C++ Interface	32
Subscribing to Gaze Data	32
Available Data	32
Time Stamp	32
Eye Position	32
Relative Eye Position	32
Gaze Point	32
Relative Gaze Point	33
Validity Code	33
Pupil Diameter	34
Data Reference	34
Gaze Data Subscription	35
C# Interface	35
Python Interface	36
C++ Interface	36
Objective-C interface	37

Calibration	37
Calibration Procedure	37
Calibration Plot	39
Calibration Buffers	39
Calibration Interface	40
C# Interface	40
Python Interface	41
C++ Interface	42
Objective-C interface	43
The Calibration class	43
C# Interface	43
Python Interface	44
C++ Interface	44
Objective-C interface	45
Clock Functionality	45
Getting the Current Time	45
The Clock class	45
C# Interface	45
Python Interface	46
C++ Interface	46
Objective-C interface	47
Synchronizing clocks with the eye tracker	47
Synchronization Process	48
Synchronization Scenarios	51
One stimuli computer and one eye tracker	51
One stimuli computer and several eye trackers	52
Two (or more) stimuli computers and one eye tracker	53
Several setups of stimuli computers and eye trackers	54
The SyncManager class	54
C# Interface	54
Python Interface	55
C++ Interface	56
Objective-C interface	56
Getting and Setting Eye Tracker Properties	57
Basic Eye Tracker Info	57
C# Interface	57
Python Interface	57

C++ Interface	58
Objective-C interface	58
Getting and Setting the Eye tracker's Frame rate	58
C# Interface	59
Python Interface	59
C++ Interface	60
Objective-C interface	60
Querying the Eye Tracker for its Head Movement Box	60
C# Interface	61
Python Interface	62
C++ Interface	62
Objective-C interface	63
Configuring the Screen (only X-Units)	63
API for Configuring the Screen	64
C# Interface	64
Python Interface	65
C++ Interface	65
Objective-C Interface	65
Extended Functionality	66
C# Interface	66
Python Interface	67
C++ Interface	67
Objective-C Interface	68
Firmware Upgrade	69
Upgrade Functions/Methods.....	69
C# Interface	69
Python Interface	70
C++ Interface	70
Objective-C Interface	71
Troubleshooting	72
Logging.....	72
Windows Error Messages	72
Error codes	73
Generic Errors.....	73
Param Stack Errors.....	73
Transport Errors.....	73
Factory Info Errors	74

Firmware Upgrade Errors	74
Eye Tracker Errors.....	75

Table of Figures and Tables

Figure 1: .NET Eyetracking Sample	12
Figure 2: The .NET 3D Eyes sample showing the eyes 3D position, gaze vectors and head movement box.....	13
Figure 3: The .NET Upgrade Sample application	13
Figure 4: Python Eyetracker Browser example for PyGTK	16
Figure 5: Python Upgrade Tool example application.....	16
Figure 6: The User Coordinate System	23
Figure 7: Gaze Point and Gaze Vector	33
Figure 8: Standard 5-point Calibration Pattern	38
Figure 9: 5-point calibration plot with combined data for both the left and the right eye.....	39
Figure 10: Synchronization procedure.....	49
Figure 11: Synchronization Sample	49
Figure 12: Standard Synchronization Scenario	51
Figure 13: Synchronization Scenario with several eye trackers	52
Figure 14: Synchronization Scenario with one eye tracker and several applications.....	53
Figure 15: Synchronization Scenario with multiple eye trackers and applications and external clock source	54
Figure 16: Schematic head movement box	61
Figure 17: Screen Configuration Schematic.....	64
Figure 18: Upgrade Procedure.....	69
 Table 1: Eye Tracker Info Properties.....	 23
Table 2: Data Validity Codes	34
Table 3: Gaze Data Reference.....	34

Tobii SDK 3.0

Introduction

This is the developers guide for the Tobii SDK 3.0. It contains information on how to get started developing applications for Tobii eye trackers.

New and Noteworthy

- Access to the full spectrum of Tobii eye tracking hardware features:
 - More gaze data content such as 3D coordinates for eye position and gaze point, which enables easier integration in 3D environments and development of gaze angle based eye movement filters
 - Possibility to both set and retrieve settings for the calibration plane from the eye tracker which enables finer integration of SDK application and hardware as well as visualization of the current setup
- Full access to and control over the high-resolution clock synchronization between stimuli computer and eye tracker which makes the synchronization procedure more transparent compared to the old implementation.
- Access to the firmware upgrade functionality which enables third-party applications to perform firmware upgrades (which in term enables applications to depend on very recent features).
- Applications using the Tobii SDK 3.0 do not require administrative privileges, greatly increasing security and user experience over the previous versions of the SDK.

High Level Changes

1. **Unlike previous versions of the SDK, the new SDK does not use Microsoft COM technology:**
 - a. This enable SDK users to chose for themselves which version of the SDK they want to deploy with their application without affecting other applications using another version of the SDK running on the same computer
 - b. The previous SDK versions have only worked on computers with a Microsoft Windows platform due to it using Microsoft COM technology. The new SDK version is compatible with other system platforms as well.
2. **The new SDK will be available on several platforms:** the new SDK is currently available for Linux and Windows. More platforms will be supported in the future.
3. **The new SDK is deployed on a per-application basis.** This means that each application developer can choose which version of the SDK they want to deploy and that several different versions of the SDK can be in use at the same time by different applications on the same computer.
4. **The new SDK does not contain any graphical components:** instead of supplying binary components tied to a specific technology, all the graphical components have been supplied as example code
5. **The new SDK has only one API** (whereas the old SDK had a low-level and high-level API) per programming language and all functionality will be available for all languages
6. **The new SDK is asynchronous/event-based which will make it easier to write responsive GUI applications.**
7. **The new SDK is a complete rewrite:** Applications will have to be rewritten to use the new SDK. As it also includes extra eye tracking functionality; a specific firmware version is required. The new firmware is however completely backwards-compatible with older SDK versions so that existing applications will continue to work without modifications.

8. The new SDK will cover the complete functionality of Tobii's Eye Trackers, which will give users the ability to create complete eye tracking solutions.

What is included in this release?

This release contains support for the following platforms, languages and architectures:

	Windows 32bit	Windows 64bit	Ubuntu 10.04 LTS 32bit	Ubuntu 10.04 LTS 64bit	Mac OS X 32bit & 64bit
.NET	Yes	Yes	Yes (Mono)	Yes (Mono)	Yes
Python 2.6	Yes	Yes	Yes	Yes	No
C++	Yes	Yes	Yes	Yes	Yes
Cocoa	No	No	No	No	Yes

Eye tracker Hardware Support

Compared to earlier versions, the Tobii SDK 3 poses additional requirements on the firmware of the eye tracker and as such not all eye trackers will work out of the box with this SDK release. Most Tobii Eye Tracker models will be supported in the final release.

Supported Eye trackers

The following Tobii Eye Trackers are supported out-of-the-box in the Tobii SDK 3.0 RC 1 release: **Tobii X60, Tobii X120, Tobii T60, Tobii T120, Tobii T60 XL, Tobii TX300, and Tobii IS-1.**

Not supported models are: **Tobii 1750, Tobii X50, Tobii 2150, Tobii 1740 (D10), Tobii P10, C-Eye, and PCEye.**

Support for **Tobii 1750, Tobii X50, and Tobii 2150** models will be added in a later firmware release.

A firmware upgrade in order to support the Tobii SDK 3 will never break compatibility with older SDKs.

Applications written using older SDKs will keep working with all above listed models; and applications using the new SDK and applications using the old SDK can run side-by-side without problems.

Firmware Requirements

To use the Tobii SDK 3.0 with any of the supported Tobii Eye Tracker models, the **firmware** needs to be upgraded. In general, always make sure to use the *latest available* firmware for your eye tracker model. Firmwares that support the Tobii SDK 3.0 are:

For Tobii X60, X120, T60 and T120, use: **tx-firmware-2-0-0.tobiipkg** (or higher)

For Tobii TX60 XL use: **xl-firmware-2-0-0.tobiipkg** (or higher)

And for Tobii TX300 use: **mfp-firmware-1-0-0.tobiipkg** (or higher)

Release Deliverables

This release is delivered as 6 different files depending on your platform and architecture:

Windows

- **tobiisdk-[Version number]-Win32.zip**: for 32-bit Windows (.Net, Python 2.6 and C++)
- **tobiisdk-[Version number]-x 64**: for 64-bit Windows (.Net, Python 2.6 and C++)

Ubuntu 10.04

- **libtobii-sdcore_[Version number]_i386.deb**: for 32-bit Ubuntu 10.04
- **python-tobii-sdcore_[Version number]_i386.deb**: for 32-bit Ubuntu 10.04 (Python 2.6 modules)
 - Requires: **libtobii-sdcore_[Version number]_i386.deb**

- **libtobii-sdkcore_[Version number]_amd64.deb**: for 64-bit Ubuntu 10.04
- **python-tobii-sdkcore_[Version number]_amd64.deb**: for 64-bit Ubuntu 10.04 (Python 2.6 modules)
 - Requires: **libtobii-sdkcore_[Version number]_amd64.deb**

For .NET development on Linux (using mono) the appropriate Windows zip is also required.

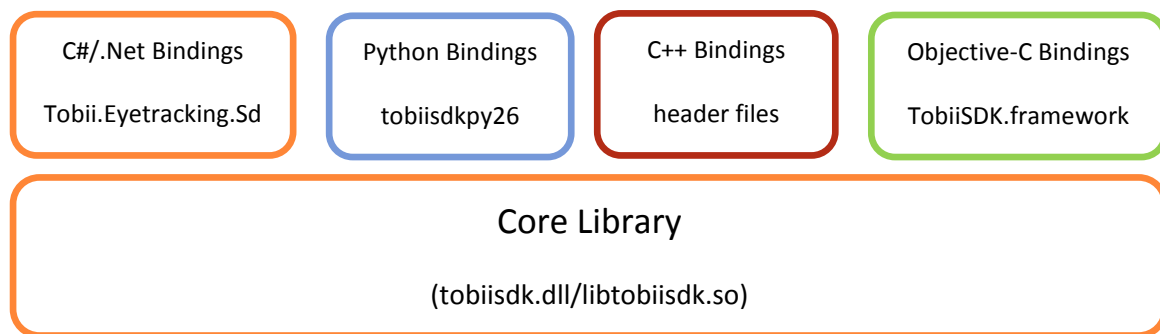
Mac OS X

- TobiiSDK.dmg: Universal binaries for 32-bit and 64 bit Mac OS X

For .NET development on Mac OS X (using mono) the appropriate Windows zip is also required.

SDK Components

The Tobii SDK consists of a core library with a C-style interface. On top of the core library we have created language specific bindings that make it possible to access the eye tracker functionality from several different programming languages.



Getting Started

Getting Started on Windows

Before you unpack the SDK

The SDK currently requires dnssd.dll from Bonjour to be installed. Bonjour is an implementation of the zeroconf standard made by Apple. This standard describes how to automatically find devices connected to a local network without having to do any advanced configurations.

Bonjour can be downloaded from Apple:

<http://support.apple.com/kb/dl999>

Bonjour, the Bonjour logo, and the Bonjour symbol are trademarks of Apple Computer, Inc.

In order to get started unpack the desired ZIP-variant (Win32 or Win64). The files should unpack into the following folder structure:

- Win32 or Win64
 - C++
 - Includes

- Binaries
- Net
 - Binaries
 - Examples
 - Binaries
 - Basic Eyetracking Sample
- Python26
 - Modules
 - Examples
 - PyGtk

Here is a short description of the folders' contents:

1. **C++/Includes** contains headers for C and C++
2. **C++/Binaries** contains dll and lib files needed to compile and redistribute applications written using C++
3. **Net/Binaries** contains all the redistributable files required for the .NET SDK
4. **Net/Examples/Binaries** directory contains compiled binaries for the examples together with another copy of the redistributable files so that they can be started directly from within that directory.
5. **Python26/Modules** contains the python modules
6. **Python26/Examples/PyGtk** contains examples written in Python and using the PyGTK toolkit. In order to run them, PyGtk must be installed separately. This covered in detail later in this document.

Running the .NET Examples

You can run the "Basic" example by double-clicking on `Net/Examples/Binaries/SDK Basic Eyetracking Sample.exe`. This should bring up an application that should look like the image seen below (see Figure 1). The full source code of this sample is included in the subfolder `Net/Examples/Basic Eyetracking Sample` and shows how to use the new eye tracking programming interface from C# and Windows Forms. It illustrates how to find eye trackers on the network, connect to a specific eye tracker, perform a calibration and subscribe to data from the eye tracker.

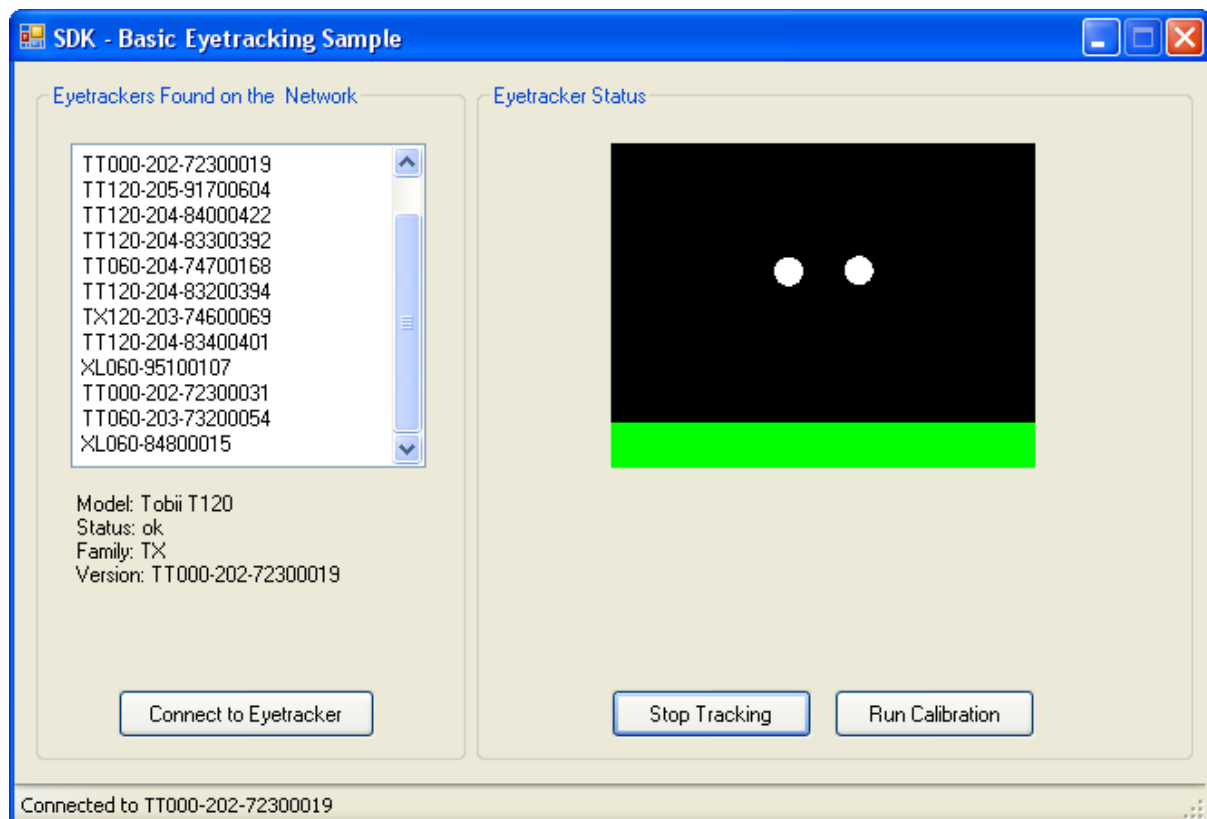


Figure 1: .NET Eyetracking Sample

In the folder `Net/Examples/Binaries` you will also find an application called `3DEyes.exe`. This application is a demonstration of the 3D information available in the new SDK. Having access to true eye position and gaze points in space makes it possible to create compelling visualizations.

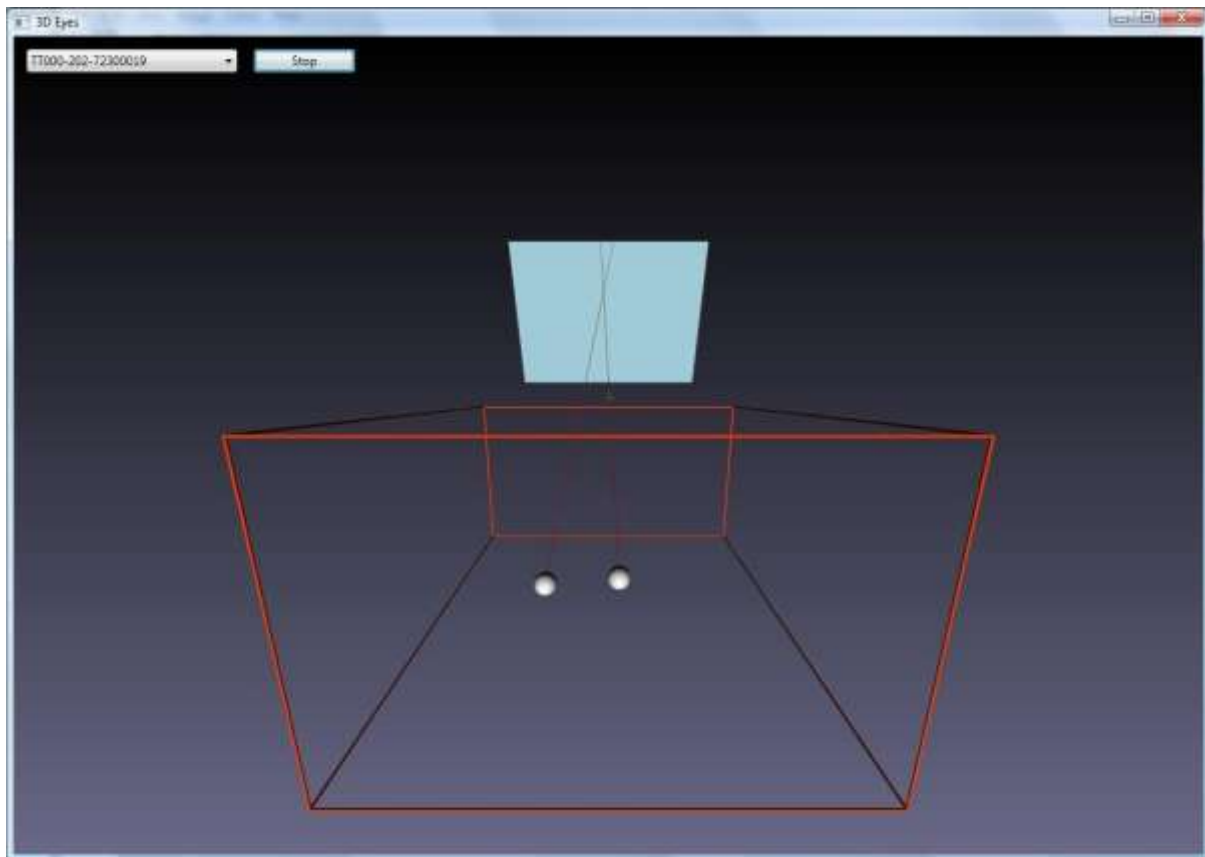


Figure 2: The .NET 3D Eyes sample showing the eyes 3D position, gaze vectors and head movement box

Also included is a small sample application that demonstrates the firmware upgrade capabilities. This application is called `UpgradeSample.exe` and can be found in the same folder.

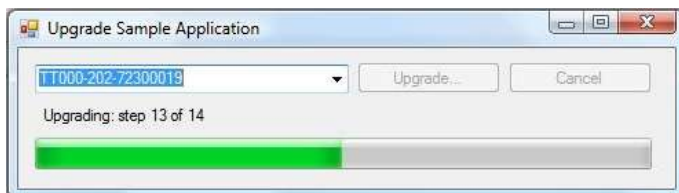


Figure 3: The .NET Upgrade Sample application

Setting up the Python Environment

After installing the appropriate Python 2.6 variant from <http://www.python.org/> you need to ensure that the Tobii SDK modules can be found on the Python path. For convenience you might also want to add the path of `python.exe` to the path. The search path for Python modules is controlled by the environment variable `PYTHONPATH` (just like the search path for executables is controlled by `PATH`). To set these environment variables, use the following procedure: If you unpacked the beta ZIP to, e.g., `C:\Users\<name>\Desktop\[FolderName]` and Python is installed in `C:\Python26\` (which is the default location), open up a command prompt and enter the following commands:

- `set PATH=%PATH%;C:\Python26\`
- `set PYTHONPATH=C:\Users\<name>\Desktop\[FolderName]\Win<32/64>\Python26\Modules`

You can also set these environment variables permanently in the Windows control panel.

- Once the environment variables are configured, the examples can be run from the command prompt by entering the following commands: `cd C:\Users\<name>\Desktop\[FolderName]\Win<32/64>\Python26\Examples`
- `python test-EyetrackerBrowser.py` (or any other example)

The subdirectory `Python26\Examples\PyGtk` contains python examples which use the PyGTK graphical toolkit. The PyGTK graphical toolkit is not installed by default on Windows. PyGTK can be obtained from the following URL: <http://www.pygtk.org/>.

Getting Started on Ubuntu Linux 10.04 LTS

Depending on whether you are running a 32bit or 64bit variant of Ubuntu you will need to choose the correct set of packages to install:

For 32bit please install:

- **libtobii-sdkcore_[Version number]_i386.deb**
- **python-tobii-sdkcore_[Version number]_i386.deb**

For 64bit please install:

- **libtobii-sdkcore_[Version number]_amd64.deb**
- **python-tobii-sdkcore_[Version number]_amd64.deb**

On Ubuntu 10.04 LTS you can install the packages by double-clicking on the respective .deb file. If you want to install via the terminal write:

- `sudo dpkg --install libtobii* python-tobii*`

The command should be executed from within the directory where you have saved the two .deb files. This command will install the python modules into the python module path. The package `libtobii-sdkcore_[Version number]_*.deb` installs a private, shared object which is used by the python module.

In order to test whether the installation succeeded, open up a terminal and enter the following commands (the commands following '`>>>`' occur inside the Python interactive console):

```
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import tobii.sdk
>>> tobii.sdk.init()
>>>
```

If the commands execute successfully, i.e. without error messages, the SDK has been installed correctly.

Prerequisites on Ubuntu 10.04 LTS

The SDK depends on Avahi to provide MDNS¹ and DNS-SD² support. Avahi basically provides the same functionality as Bonjour does on Windows. Avahi should be installed automatically when you install the .deb packages.

On some Ubuntu machines Avahi does not start when it detects a unicast DNS **.local** domain. However, this check for unicast DNS .local domains seems to give false-positives sometimes. In these cases a workaround is

¹ Multicast DNS

² DNS Service Discovery

required in order to disable the check³. In order to disable the check for unicast .local domain, create the file: `/etc/default/avahi-daemon` with the contents `AVAHI_DAEMON_DETECT_LOCAL=0`, and then restart the `avahi-daemon`.

Running the examples on Ubuntu Linux 10.04 LTS

Running the Python Examples

The package `python-tobii-sdkcore` installs examples in `/usr/share/doc/python-tobii-sdkcore/examples` and `/usr/share/doc/python-tobii-sdkcore/examples/gtk`. To run the examples that do not require the GTK toolkit, use the following command and replace the filename given below by the filename of the example you want to run: `$ /usr/share/doc/python-tobii-sdkcore/examples/test-EyetrackerBrowser.py`

Running the Python GTK Examples

The package `python-tobii-sdkcore` installs GTK (the GUI toolkit used by Gnome) examples in `/usr/share/doc/python-tobii-sdkcore/examples/gtk`. They can, however, not be executed directly as they must be extracted first (they are automatically gzipped by the Ubuntu packaging tools). This is because these examples have been compressed into gzip-files. To unzip all the examples at once and make them executable, use the following command:

```
$ for EXAMPLE in /usr/share/doc/python-tobii-sdkcore/examples/gtk/*.py; do TARGET="$(basename $EXAMPLE .py.gz).py"; zcat $EXAMPLE > $TARGET; chmod +x $TARGET; done
```

Please note that the entire command must be entered before it is executed.

Once the examples have been extracted, they can be executed by entering a command as seen below. The command seen here will execute the example 'EyetrackerBrowser.py'.

```
$ ./EyetrackerBrowser.py
```

If entered correctly, 'EyetrackerBrowser.py' should launch a window which looks like the image below (see Figure 4). This sample illustrates how to find eye trackers on the network, connect to a specific eye tracker, perform a calibration and subscribe to data from the eye tracker.

³ <https://bugs.launchpad.net/ubuntu/+source/avahi/+bug/327362>

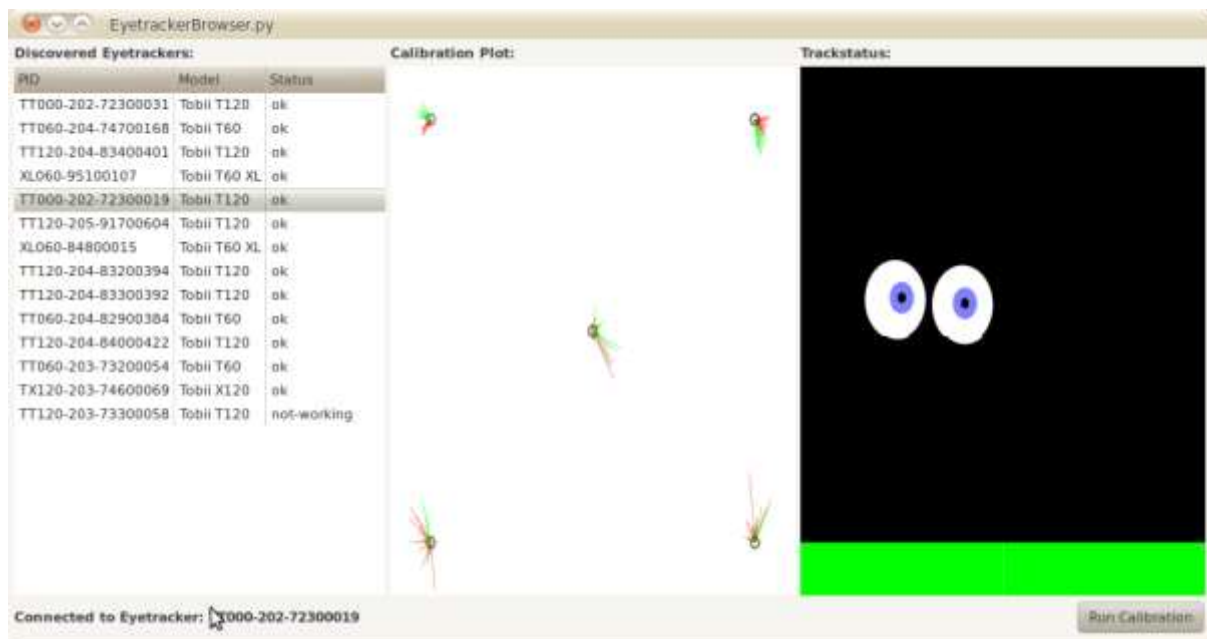


Figure 4: Python Eyetracker Browser example for PyGTK

The Python example collection also contains a small sample application that demonstrates the upgrade functionality provided by the sdk. The sample can be found in the file UpgradeTool.py.



Figure 5: Python Upgrade Tool example application

Running the .NET Examples on Ubuntu 10.04 LTS

Because the .NET is supported on Linux by means of Mono⁴, you can also use .NET languages to write SDK applications on Linux.

In order to run the .NET examples you must have installed the package **libtobii-sdkcore_[Version number]_amd64.deb** or **libtobii-sdkcore_[Version number]_i386.deb**, depending on your architecture. To run the samples you then have to get the files from the corresponding Windows zip-files. These are named **tobiisdk-[Version number]-win32.zip** or **tobiisdk-[Version number]-x64.zip**, depending on the platform. .

The following commands executes the Sample.exe example:

```
$ cd Net/Examples/Binaries
$ chmod +x SDK\ Basic\ Eyetracking\ Sample.exe
$ ./SDK\ Basic\ Eyetracking\ Sample.exe
```

The sample program that is executed by the command above should look the same as the image for the sample program in the Windows .NET section in Figure 1.

⁴ <http://www.mono-project.com/>

The sample project can be opened, compiled and started from within MonoDevelop on Ubuntu 10.04 LTS.

Getting Started on Mac OS X

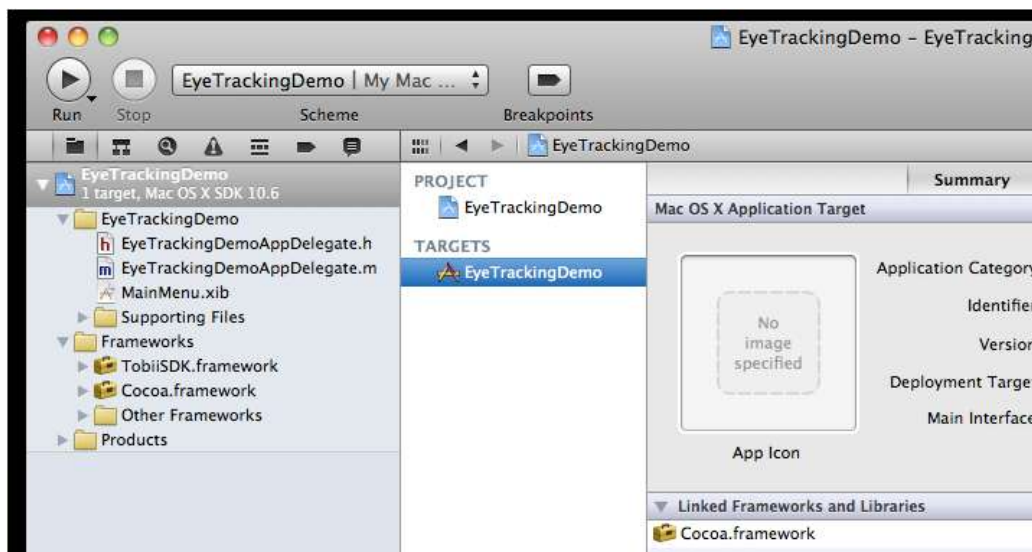
The TobiiSDK is packaged in an Apple Framework to simplify writing programs in Objective-C using the powerful Cocoa-framework.

Installing

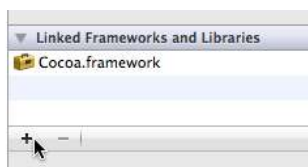
To install the TobiiSDK framework, mount the dmg file and drag-and-drop the TobiiSDK.framework folder to the default location /Library/Frameworks/ or any other suitable folder.

Referencing TobiiSDK

To use the TobiiSDK from your XCode 4 project select your application target from the Project Navigator.



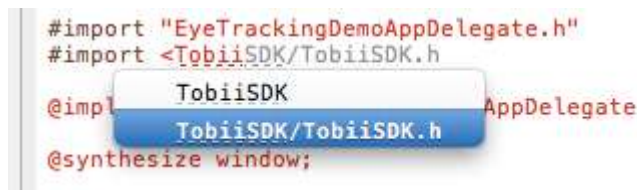
In the summary tab there is a section called “Linked Frameworks and Libraries”. Press the plus-sign to add a new framework.



Select the “TobiiSDK.framework” and press “Add”. The TobiiSDK.framework now appears under “Linked Frameworks and Libraries”



You can now include the master header file.



```
#import "EyeTrackingDemoAppDelegate.h"
#import <TobiiSDK/TobiiSDK.h>
@implementation AppDelegate
    @synthesize window;
```

With the master header file from the framework you can use the types from the framework. All classes included in the TobiiSDK framework has a TET-prefix to improve discoverability.



```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    [TETEyeTrackerBrowser
}
TETExtension
TETExternalClock
TETEyeTracker
TETEyeTrackerBrowser
TETEyeTrackerFactory
TETEyeTrackerInfo
```

Using TobiiSDK from mono

To use the TobiiSDK from mono copy the dylib from inside the framework bundle to alongside the mono executable with the name `libtobiisdk.dylib`. For example to run the “Basic Eyetracking Sample” from the windows zip, open a terminal and run the following command

```
cp /Library/Frameworks/TobiiSDK.framework/TobiiSDK <path to folder with SDK
Basic Eyetracking Sample.exe>/libtobiisdk.dylib
```

Developing SDK Applications

Library Initialization

Before you can call any function in the library it must first be initialized. To do this, add the following lines somewhere in your application startup code:

Language	
C#	<code>Tobii.Eyetracking.Sdk.Library.Init()</code>
Python	<code>tobii.sdk.init()</code>
C++	<code>tobii::sdk::cpp::init_library()</code>
Objective-C	<code><automatic - no call needed></code>

This call will perform the necessary initializations of the SDK library.

Thread Handling in the Different Languages

Since the eye tracking SDK must be able to handle data that arrives to the client in an asynchronous fashion, the core is running multiple threads that handle reads and writes to and from the eye tracker hardware. This has some implications for applications that use the SDK. The multithreading is handled differently by the supported languages since different programming languages have different levels of support for multithreading.

C#/.Net Specifics

The Microsoft .Net framework has built-in support for advanced multithreading. This has made it possible to hide a lot of the complexity involved in the .Net language bindings. You will notice that all classes that expose events to the client application takes a parameter of type `EventThreadingOptions` in the constructor. This is an enum type that indicates how events should be delivered to listeners and can be interpreted as follows:

EventThreadingOptions value	Description
CallingThread	All events are delivered on the thread that calls the constructor. In a single threaded application this means that all events appears on the main thread.
BackgroundThread	All events are delivered on a background thread. When this option is used, care must be taken to ensure that this thread doesn't manipulate the user interface which would cause a cross thread exception to occur.

Python Specifics

In the Python language bindings, all events are delivered on a background thread. This thread is provided by the user in the form of a `MainloopThread` object. The `MainloopThread` object is created by the client application and provided as an argument to all the constructors that need a mainloop instance. Typically you create an instance of a `MainloopThread` and keep it alive for rest of the application's lifetime. When the application exits you must make sure to call `MainloopThread.stop()` to shut down all background threads. Another important thing to keep in mind when developing Python applications is that you never should call back into the SDK from the background thread or deadlocks can occur. Instead you should direct all events from the background thread to the main thread. GUI frameworks typically offer this functionality, since the user interface generally can be manipulated only from the main thread. In, e.g., the Pygtk library, the `glib_idle_add()` function can be used to switch threads to the main thread. The Python sample applications show both how to use the `MainloopThread` class and how to marshal calls from background threads to the main UI thread.

Namespace
<code>tobii.sdk.mainloop</code>

Signature
<code>class MainloopThread(object) :</code>

Constructors	Description
<code>def __init__(self, mainloop=None, delay_start=False) :</code>	Creates a new mainloop instance.

Methods	Description
<code>def start(self) :</code>	Starts the mainloop.
<code>def stop(self) :</code>	Stops the mainloop.

C++ Specifics

The C++ language bindings are similar to the Python bindings, in that all events are delivered on a background thread and many of the class constructors take an instance of a mainloop object as an argument. Typically the SDK client application creates an instance of this type and keep it alive for the duration of the application. The mainloop class is not attached to a thread when it is created. Hence, the application has to first create a thread and then call the `mainloop::run()` method. This call will block until someone calls `mainloop::quit()`. When that happens the thread will return from `mainloop::run()` and continue the execution. As in the Python case it is important to not call back into the SDK from the background thread since that may cause deadlocks. Instead, events should be marshaled to the main thread (or possibly another thread). This can usually be accomplished by using a platform specific library, .e.g., MFC on Windows or gtkmm on Linux.

Namespace
<code>tobii::sdk::cpp</code>

Signature
<code>class mainloop</code>

Constructors	Description
<code>Mainloop</code>	Creates a new mainloop instance.

Methods	Description
<code>void run()</code>	Starts the mainloop. This call is blocking and should be called on a background thread. All asynchronous events will be delivered on this thread.
<code>void quit()</code>	Tell the mainloop thread to stop doing work and return execution from the blocking call to <code>run()</code> . This is typically done when the

application exits.

Objective-C Specifics

The Cocoa framework has built in support for advanced multithreading much like .NET. Objects emitting events via a delegate, i.e. `TETEyetrackerBrowser` and `TETEyetracker`, have an associated thread object (`NSThread`). Delegate calls will be called on that thread. It's also possible to pass `nil` as target thread. In this case the event message on the delegate will be sent from a background thread created by the SDK. Other messages sent to SDK objects may cause dead-lock while inside the message handler when this feature is disabled by passing `nil`.

```
[[TETEyetrackerBrowser alloc] initWithTargetThread:nil]
```

The default is to use the `[NSThread mainThread]` message to get the thread to post events to.

```
[[TETEyetrackerBrowser alloc] init]
```

Handling Errors and Exception

All language bindings except Objective-C use exceptions to notify the user when an error occurs. The SDK library contains an `EyetrackerException` that is available for the different languages. This exception contains an error code that further specifies what kind of error that occurred. The different error codes and their interpretations are summarized in the appendix of this document.

The EyetrackerException class

C# Interface

Namespace	
<code>Tobii.Eyetracking.Sdk.Exceptions</code>	

Signature	
<code>public class EyetrackerException : Exception</code>	

Constructors	Description
<code>EyetrackerException(Int32 code)</code>	Creates an instance from a specific error code.
<code>EyetrackerException(Int32 code, string message)</code>	Creates an instance from a specific error code and a message.
<code>EyetrackerException(Int32 code, string message, Exception innerException)</code>	Creates an instance from a specific error code, a message and an inner exception.

Properties	Description
<code>Int32 ErrorCode { get; }</code>	Gets the error code value for this instance.

Python Interface

Namespace	
<code>tobii.sdk.basic</code>	
Signature	
<code>class EyetrackerException(Exception):</code>	
Constructors	Description
<code>def __init__(self, error):</code>	Creates an instance from a specific error code.
Properties	Description
<code>self.error</code>	Gets the error code value for this instance.

C++ Interface

Namespace	
<code>tobii::sdk::cpp</code>	
Signature	
<code>class eyetracker_exception</code>	
Constructors	Description
<code>eyetracker_exception(error_code_t code, const std::string& message)</code>	Creates an instance from a specific error code and a message.
Methods	Description
<code>error_code_t get_error_code()</code>	Gets the error code value for this instance.

Cocoa error-handling

In the Objective-C binding the `NSError` class is used to denote errors. Some messages may take an optional for `NSError**` for example a `[EyeTrackerBrowser start]` message. If the message fail this pointer is populated with an `NSError` object describing the error.

Also for errors may be reported via a delegate. This is the case for the `TETEyeTracker` class. If a message to an instance of this class fails, the `eyetracker:didFailWithError: message` is sent to the delegate.

Some Notes on Coordinate Systems

All the data available from Tobii Eye Trackers that describe space coordinates are given in something called the *User Coordinate System* or *UCS* for short. This coordinate system is a millimeter based eye tracker fixed system with origin at the center of the frontal surface of the eye tracker. The coordinate axes are oriented as follows:

the x-axis points horizontally towards the user's right, the y-axis points vertically towards the users up and the z-axis points towards the user, perpendicular to the filter surface.



Figure 6. The User Coordinate System on a T60/T120 eye tracker

Browsing for available eye trackers

Finding Eye Trackers

The Tobii SDK contains functionality to automatically find Tobii Eye Trackers connected to a local area network. This functionality is built on *Zeroconf*, a set of techniques that allows programs to publish and discover services running on a local network without the need to do any specific configurations.

The browser functionality is available through the *EyetrackerBrowser* class, which is an implementation of the well known *Observer design pattern*. Instances of this class can notify any listeners when one of the following events occur:

- An eye tracker appears on the network
- An eye tracker disappears from the network
- An eye tracker has one of its properties changed

Eye tracker Properties

Every eye tracker has a collection of properties that is announced on the local network. These are summarized in the following table:

Table 1: Eye Tracker Info Properties

Property	Description
Generation	A string describing the product generation. The currently existing values are "TX" (Tobii T60, Tobii T120, Tobii X60, Tobii X120), "XL" (Tobii T60 XL) and "MSP" (TX300), but more will be added in the future
Model	A string describing the product model, e.g. "Tobii T60"
Version	A string that represents the firmware version of the specific unit. Usually on the format "1.2.3", but other variations exist.
Product ID	A string that describes the product ID, e.g. "TT060-203-52300012". This property is guaranteed to be unique among all Tobii products and can be used to identify a specific unit.
Status	A status string that describes the current status of a specific unit. Can have the

	following values: <ul style="list-style-type: none"> • “ok” – unit is working as expected • “not-working” – unit is up and running but the eye tracker firmware is not working as expected • “upgrading” – unit is installing a firmware upgrade • “error” – indicates a serious error. The unit is malfunctioning in some way
Given Name	A string describing the unit name. The name property can be changed by any SDK client and is of no vital importance for the operations of the eye tracker.

The *Status* property is very important as an eye tracker will only accept connecting clients if it is in the “ok” state. The eye tracker properties are exposed in the form of an *EyetrackerInfo* class. When an *EyetrackerBrowser* object notifies its listeners, an instance of the *EyetrackerInfo* is always passed as a parameter so that the listeners can keep track of all eye trackers found on the local network.

The EyetrackerBrowser class

C# Interface

Namespace
<code>Tobii.Eyetracking.Sdk</code>

Signature
<code>class EyetrackerBrowser : IEyetrackerBrowser</code>

Constructors	Description
<code>EyetrackerBrowser()</code>	Creates an <i>EyetrackerBrowser</i> instance
<code>EyetrackerBrowser (EventThreadingOptions opt)</code>	Creates an <i>EyetrackerBrowser</i> instance using the specific threading option for events.

Properties	Description
<code>bool IsStarted { get; }</code>	Gets a value which indicates if the eye tracker has been started or not.

Methods	Description
<code>void Start()</code>	Starts the browser. Subscribers will not get any event notifications until the browser is started.
<code>void Stop()</code>	Stops the browser.

Events	Description
<code>event EventHandler<EyetrackerInfoEventArgs> EyetrackerFound;</code>	This event is fired when an eye tracker appears on the network.

<code>event EventHandler<EyetrackerInfoEventArgs> EyetrackerUpdated;</code>	This event is fired when an existing eye tracker has one or several of its properties updated.
<code>event EventHandler<EyetrackerInfoEventArgs> EyetrackerRemoved;</code>	This event is fired when an eye tracker disappears from the network.

Python Interface

Namespace	
<code>tobii.sdk.browsing</code>	
Signature	
<code>class EyetrackerBrowser(object):</code>	
Constructors	Description
<code>def __init__(self, mainloop, callback, *args):</code>	Creates and starts an EyetrackerBrowser instance. Event notifications will be sent to the provided callback.
Methods	Description
<code>def stop(self):</code>	Stops the browser.

C++ Interface

Namespace	
<code>tobii::sdk::cpp::discovery</code>	
Signature	
<code>class eyetracker_browser</code>	
Constructors	Description
<code>eyetracker_browser(const mainloop& mainloop)</code>	Creates an EyetrackerBrowser instance
Methods	Description
<code>void start()</code>	Starts the eye tracker browser. Subscribers will not get any event notifications until the browser is started.
<code>void stop()</code>	Stops the eye tracker browser.

```
connection add_browser_event_listener(
    const browser_event::slot_type& listener)
```

Adds a browser callback event listener to this eye tracker browser instance.

Objective-C interface

Signature

```
@class TETEyeTrackerBrowser
```

Messages	Description
<code>-(id) init</code>	Initializes the eye tracker browser. Events from this browser will be posted on the main thread.
<code>-(id) initWithTargetThread: (NSThread*) thread</code>	Initializes the eye tracker browser. Events from this browser will be posted on the given thread. If nil is passed events will not be reposted on another thread.
<code>-(BOOL) start: (NSError**) errorOrNotUsed</code>	Starts the browsing. Returns YES on success and NO on failure.
<code>-(void) stop</code>	Stops the browsing.
<code>@property (readwrite, assign) id<TETEyeTrackerBrowserDelegate> delegate</code>	Gets or sets the delegate to which messages are sent when an eye tracker is found, lost or updated.

Signature

```
@class TETEyeTrackerBrowserDelegate
```

Messages	Description
<code>-(void) eyeTrackerBrowser: (TETEyeTrackerBrowser*) browser didFindEyeTracker: (TETEyeTrackerInfo*) eyeTrackerInfo;</code>	Called on the delegate when an eye tracker is found.
<code>-(void) eyeTrackerBrowser: (TETEyeTrackerBrowser*) browser didLoseEyeTracker: (TETEyeTrackerInfo*) eyeTrackerInfo;</code>	Called on the delegate when an eye tracker is lost.
<code>-(void) eyeTrackerBrowser: (TETEyeTrackerBrowser*) browser didUpdateEyeTracker: (TETEyeTrackerInfo*) eyeTrackerInfo;</code>	Called on the delegate when an eye tracker is updated.

The EyetrackerInfo class

C# Interface

Namespace

```
Tobii.Eyetracking.Sdk
```

Signature

```
public class EyetrackerInfo : IEquatable<EyetrackerInfo>
```

Constructors	Description
(None)	It is not possible to create instances of the EyetrackerInfo class directly.

Properties	Description
<code>string ProductId { get; }</code>	Gets the product id property as a string
<code>string GivenName { get; }</code>	Gets the given name property as a string
<code>string Model { get; }</code>	Gets the model property as a string
<code>string Generation { get; }</code>	Gets the generation property as a string
<code>string Version { get; }</code>	Gets the version property as a string
<code>string Status { get; }</code>	Gets the status property as a string. This will be any of the values "ok", "not-working", "upgrading" or "error".
<code>FactoryInfo FactoryInfo { get; }</code>	Gets a FactoryInfo object from this EyetrackerInfo. This object is used when connecting to an eye tracker.

Python Interface

Namespace
<code>tobii.sdk.browsing</code>

Signature
<code>class EyetrackerInfo(object):</code>

Constructors	Description
(None)	It is not possible to create instances of the EyetrackerInfo class directly.

Properties	Description
<code>@property def product_id(self):</code>	Gets the product id property as a string
<code>@property def given_name (self):</code>	Gets the given name property as a string
<code>@property def model(self):</code>	Gets the model property as a string
<code>@property def generation(self):</code>	Gets the generation property as a string

<code>@property def firmware_version(self):</code>	Gets the version property as a string
<code>@property def status(self):</code>	Gets the status property as a string. This will be any of the values "ok", "not-working", "upgrading" or "error".
<code>@property def factory_info(self):</code>	Gets a FactoryInfo object from this EyetrackerInfo. This object is used when connecting to an eye tracker.

C++ Interface

Namespace
<code>tobii::sdk::cpp::discovery</code>

Signature
<code>class eyetracker_info</code>

Constructors	Description
(None)	It is not possible to create instances of the EyetrackerInfo class directly.

Methods	Description
<code>std::string get_product_id()</code>	Gets the product id property as a string
<code>std::string get_given_name()</code>	Gets the given name property as a string
<code>std::string get_model()</code>	Gets the model property as a string
<code>std::string get_generation()</code>	Gets the generation property as a string
<code>std::string get_version()</code>	Gets the version property as a string
<code>std::string get_status()</code>	Gets the status property as a string. This will be any of the values "ok", "not-working", "upgrading" or "error".

Objective-C interface

This class also implements `isEqual:other` and `hash` messages so that they can easily be used in e.g. a `NSSet`.

Signature
<code>@class TETEyetrackerInfo</code>

Messages	Description
----------	-------------

- (NSString*) productId	Gets the product id property as a string
- (NSString*) givenName	Gets the given name property as a string
- (NSString*) model	Gets the model property as a string
- (NSString*) generation	Gets the generation property as a string
- (NSString*) version	Gets the version property as a string
- (NSString*) status	Gets the status property as a string. This will be any of the values "ok", "not-working", "upgrading" or "error".

Connecting to an Eye Tracker

After finding eye trackers on the local network, the next step is to establish a connection. The connection is represented by the Eyetracker class (in the C#/.Net API this is an interface), which exposes everything you can do with an eye tracker.

C# Interface

Namespace
Tobii.Eyetracking.Sdk

Class/Interface
EyetrackerFactory

Methods	Description
<code>static IEyetracker CreateEyetracker(EyetrackerInfo trackerInfo)</code>	Establishes an eye tracker connection to the unit specified by the EyetrackerInfo parameter.
<code>static IEyetracker CreateEyetracker(EyetrackerInfo trackerInfo, EventThreadingOptions threadingOptions)</code>	Establishes an eye tracker connection to the unit specified by the EyetrackerInfo parameter.
<code>static IEyetracker CreateEyetracker(FactoryInfo factoryInfo)</code>	Establishes an eye tracker connection to the unit specified by the FactoryInfo parameter.
<code>static IEyetracker CreateEyetracker(FactoryInfo factoryInfo, EventThreadingOptions threadingOptions)</code>	Establishes an eye tracker connection to the unit specified by the EyetrackerInfo parameter.
<code>IEyetracker CreateEyetracker(string ipAddress, int serverPort, int synchronizationPort)</code>	Establishes an eye tracker connection to the unit specified by the address and port parameters. Tobii Eye Trackers use server port 4455 and sync port 4457.
<code>static IEyetracker CreateEyetracker(string ipAddress, int serverPort, int synchronizationPort, EventThreadingOptions threadingOptions)</code>	Establishes an eye tracker connection to the unit specified by the address and port parameters. Tobii Eye Trackers use server port 4455 and sync port 4457.

Python Interface

Namespace	
<code>tobii.sdk.eyetracker</code>	
Class/Interface	
<code>Eyetracker</code>	
Methods	Description
<pre>@classmethod def create_async(class, mainloop, eyetracker_info, callback, *args, **kwargs):</pre>	Establishes an eye tracker connection to the unit specified by the <code>EyetrackerInfo</code> parameter.

C++ Interface

Namespace	
<code>tobii::sdk::cpp::tracking</code>	
Class/Interface	
Free function in <code>eyetracker-factory.hpp</code>	
Methods	Description
<pre>eyetracker::pointer create_eyetracker(const discovery::factory_info& info, const mainloop& loop)</pre>	Establishes an eye tracker connection to the unit specified by the <code>EyetrackerInfo</code> parameter.

Objective-C Interface

Signature	
<code>@class TETEyeTrackerFactory</code>	
Methods	Description
<pre>+(TETEyeTracker*) createEyeTracker: (TETEyeTrackerInfo*) info</pre>	Establishes an eye tracker connection to the unit specified by the <code>info</code> parameter.
<pre>+(TETEyeTracker*) createEyeTracker: (TETEyeTrackerInfo*) info targetThread: (NSThread*) thread</pre>	Establishes an eye tracker connection to the unit specified by the <code>info</code> parameter. Events from the returned tracker will be reposted on

```
error: (NSError**) errorOrNotUsed;
```

the given thread. If nil is passed as thread events will be fired on a background thread created by the SDK. If nil is returned errorOrNotUsed contains information about the error.

Handling Connection Errors

Unlike the previous versions of the SDK, this version reports asynchronous errors, i.e. errors which do not occur while actively performing a request. The eye tracker class/interface offers a notification event that is fired when, e.g., network connectivity is lost. When this event has been received the connection should be considered broken and no attempts should be made to further use the instance.

C# Interface

Namespace

`Tobii.Eyetracking.Sdk`

Class/Interface

`IEyetracker`

Event	Description
<pre>event EventHandler<ConnectionErrorEventArgs> ConnectionError;</pre>	<p>This event is fired when the connection between the client and the eye tracker is lost. When this happens the IEyetracker instance should be considered broken and should be disposed of.</p>

Python Interface

Namespace

`tobii.sdk.eyetracker`

Class/Interface

`Eyetracker`

Event	Description
<p><code>OnError</code></p>	<p>This event is fired when network connectivity is lost. When this happens the Eyetracker instance should be considered broken and should be disposed of.</p>

C++ Interface

Namespace	
<code>tobii::sdk::cpp::tracking</code>	

Class/Interface	
<code>Eyetracker</code>	

Methods	Description
<code>connection</code> <code>add_connection_error_listener(const connection_error_event::slot_type& listener)</code>	This event is fired when network connectivity is lost. When this happens the eye tracker instance should be considered broken and should be disposed of.

Subscribing to Gaze Data

Available Data

The possibility to get gaze data is probably the most interesting functionality provided by the SDK. The SDK API makes it possible to subscribe to a stream of data that will arrive to the client application according to the eye tracker sampling frequency. Here is an outline of the data included in each sample.

Time Stamp

This is a value that indicates the time when a specific gaze data sample was sampled by the eye tracker. The value should be interpreted as a 64 bit unsigned microsecond from some unknown point in time. The source of this time stamp is the internal clock in the eye tracker hardware. A typical use case is to synchronize the gaze data stream with some other kind of data stream, e.g., information on when a certain stimuli was presented on the computer screen. How this is done is described in the time sync section in the SDK documentation.

Eye Position

The eye position is provided for the left and right eye individually and describes the position of the eyeball in 3D space. Three floating point values are used to describe the x, y and z coordinate respectively. The position is described in the UCS coordinate system.

Relative Eye Position

The relative eye position is provided for the left and right eye individually and gives the relative position of the eyeball in the headbox volume as three normalized coordinates. This data can be used to visualize the position of the eyes similar to how it is done in the Tobii track status control. This is mainly a tool to help the user position himself/herself in front of the tracker.

Gaze Point

The gaze point is provided for the left and right eye individually and describes the position of the intersection between the line originating from the eye position point with the same direction as the gaze vector and the calibration plane. This is illustrated in Figure 7 .

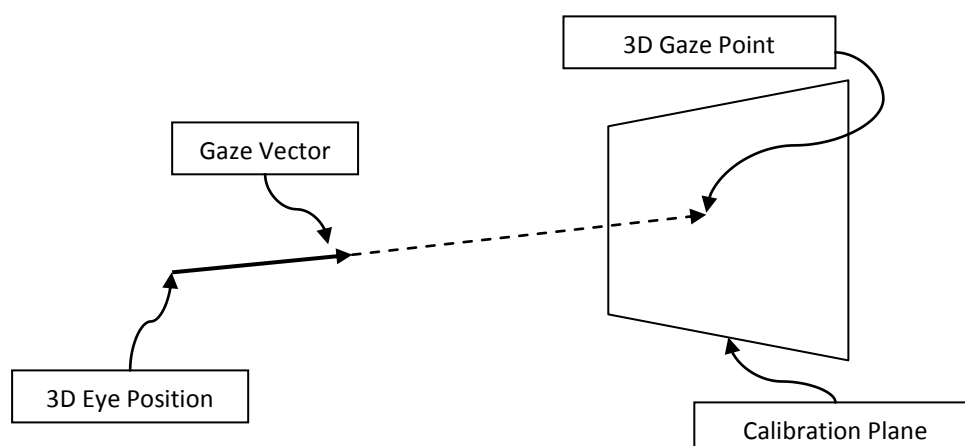


Figure 7: Gaze Point and Gaze Vector

The gaze vector can be computed by subtracting the 3D gaze point and the 3D eye position and normalizing the resulting vector.

Relative Gaze Point

The relative gaze point is provided for the left and right eye individually and corresponds to the two dimensional position of the gaze point within the calibration plane. The coordinates are normalized to [0,1] with the point (0,0) in the upper left corner from the users point of view. The x-coordinate increases to the right and the y-coordinate increases towards the bottom of the screen.

Validity Code

The validity code is an estimate of how certain the eye tracker is that the data given for an eye really originates from that eye. When the tracker finds two eyes in the camera image, identifying which one is the left and which one is the right eye is very straightforward as well as when no eyes are found at all. The most challenging case is when the tracker only finds one eye in the camera image. When that happens, the image processing algorithms try to deduce if the eye in question is the left or the right one. This is done by referring to previous eye positions, the position in the camera sensor and certain image features. The validity codes describe the outcome of this deduction. The validity codes can only appear in certain combinations. These combinations and their interpretations are summarized in this table:

Table 2: Data Validity Codes

		Right Validity Code				
Left Validity Code		0	1	2	3	4
	0	Found two eyes				Found one eye. Most probable the left eye.
	1				Found one eye. Probably the left eye.	
	2			Found one eye. The tracker cannot with any certainty determine which eye it is.		
	3		Found one eye. Probably the right eye.			
	4	Found one eye. Most probable the right eye.				No eyes found

The validity codes can be used to filter out data that is most likely incorrect. Normally it is recommended that all samples with validity code 2 or higher are removed or ignored.

Pupil Diameter

The pupil diameter data is provided for the left and the right eye individually and is an estimate of the pupil size in millimeters. The Tobii Eye Trackers try to measure the true pupil size, i.e. the algorithms take into account the magnification effect given by the spherical cornea as well as the distance to the eye.

Data Reference

This table describes the data available from our TX and XL trackers.

Table 3: Gaze Data Reference

ID	Field	Type	Description
	Time stamp	8 byte signed integer	A micro second time stamp from the eye tracker's internal clock.
	Left Eye Position	3D point given as three 8 byte floating point numbers (for x, y and z)	The 3D position of the left eye in relation to the eye tracker. Given in mm.
	Right Eye Position	3D point given as three 8 byte floating point numbers (for x, y and z)	The 3D position of the right eye in relation to the eye tracker.

		Given in mm.
Left Eye Relative Position	3D point given as three 8 byte floating point numbers (for x, y and z)	The 3D position of the left eye in relative coordinates.
Right Eye Relative Position	3D point given as three 8 byte floating point numbers (for x, y and z)	The 3D position of the right eye in relative coordinates (see next section for further description).
Left Eye Gaze Point	3D point given as three 8 byte floating point numbers (for x, y and z)	The 3D position of the left eye gaze point in mm.
Right Eye Gaze Point	3D point given as three 8 byte floating point numbers (for x, y and z)	The 3D position of the right eye gaze point in mm.
Left Eye Relative Gaze Point	2D point given as two 8 byte floating point numbers (for x, y)	The 2D position of the left eye gaze point in calibration area coordinates.
Right Eye Relative Gaze Point	2D point given as two 8 byte floating point numbers (for x, y)	The 2D position of the right eye gaze point in calibration area coordinates.
Left Eye Validity Code	4 byte unsigned integer	A value describing if the left eye was found.
Right Eye Validity Code	4 byte unsigned integer	A value describing if the right eye was found (see next section for further description).
Left Eye Pupil Diameter	8 byte floating point number	The pupil diameter of the left eye in mm
Right Eye Pupil Diameter	8 byte floating point number	The pupil diameter of the right eye in mm

Gaze Data Subscription

C# Interface

Namespace

<code>Tobii.Eyetracking.Sdk</code>

Class/Interface

<code>IEyetracker</code>

Methods	Description
<code>void StartTracking()</code>	Start subscribing to gaze data
<code>void StopTracking()</code>	Stops gaze data subscription

Events	Description
<code>event EventHandler<GazeDataEventArgs> GazeDataReceived</code>	Occurs when a gaze data sample is delivered to the client application.

Python Interface

Namespace

<code>tobii.sdk.eyetracker</code>

Class/Interface

<code>Eyetracker</code>

Methods	Description
<code>def StartTracking(self, callback, *args, **kwargs)</code>	Start subscribing to gaze data
<code>def StopTracking(self, callback, *args, **kwargs)</code>	Stops gaze data subscription

Events	Description
<code>OnGazeDataReceived</code>	Occurs when a gaze data sample is delivered to the client application.

C++ Interface

Namespace

<code>tobii::sdk::cpp::tracking</code>
--

Class/Interface

<code>Eyetracker</code>

Methods	Description
<code>void start_tracking()</code>	Start subscribing to gaze data
<code>void stop_tracking()</code>	Stops gaze data subscription
<code>connection</code> <code>add_gaze_data_received_listener(const</code> <code>gaze_data_received_event::slot_type&</code> <code>listener)</code>	Adds a listener for the gaze data event. This event occurs when a gaze data sample is delivered to the client application.

Objective-C interface

Signature
<code>@class TETEyeTracker</code>

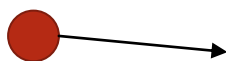
Messages	Description
<code>-(void)startTracking</code>	Starts tracking.
<code>-(void)stopTracking</code>	Stops tracking.
<code>@property (readwrite,assign)</code> <code>id<TETEyeTrackerDelegate> delegate;</code>	Gets or sets the delegate that will receive gaze data items.

Calibration

In order to compute the gaze point and gaze direction with high accuracy, the eye tracker firmware adapts the algorithms to the person sitting in front of the tracker. This adaptation is done during the calibration process, when the user is looking at points located at a known set coordinates. The calibration is initiated and controlled by the client application.

Calibration Procedure

1. The calibration of the eye tracker would typically be done as follows: A small, animated object should be shown on the screen to catch the users attention



2. When it arrives at the calibration point coordinates, let the point rest for about 0.5 seconds to give the user a chance to focus. Shrink the object to focus the gaze



3. When shrunk, tell the eye tracker to start collecting data for the specific calibration point
4. Wait for the eye tracker to finish calibration data collection on the current position
5. Enlarge the object again
6. Repeat steps 1-5 for all desired calibration points

The animation in step 1 should not be too fast, nor should the shrinking in step 2 be too fast. Otherwise the user may not be able to get a good calibration result due to the fact that she has no time to focus the gaze on the target before the eye tracker starts collecting calibration data.

The normal number of calibration points are 2, 5 or 9; more points can be used but the result will not increase significantly for more than 9 points. Usually 5 points yields a very good result and is not experienced as too intrusive by the user.

The location of the calibration points is decided by the client application. A typical calibration pattern for 5 points is:

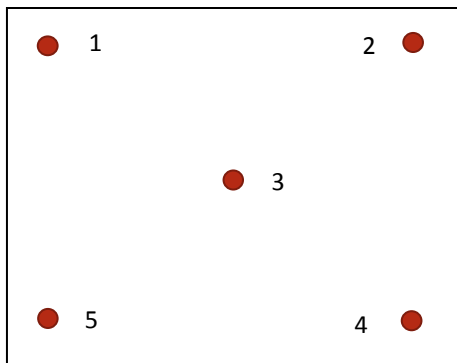


Figure 8: Standard 5-point Calibration Pattern

1. (0.1, 0.1)
2. (0.9, 0.1)
3. (0.5, 0.5)
4. (0.9, 0.9)
5. (0.1, 0.9)

Note that all points are given in normalized coordinates in such a way that (0.0, 0.0) corresponds to the upper left corner and (1.0, 1.0) corresponds to the lower right corner of the calibration area. When choosing the calibration points it is important to consider the following:

- The calibration points should span an area that is as large or larger than the area where the gaze controlled application or the stimuli is to be shown in order to ensure good interaction.
- The calibration points must be positioned within the area that is trackable by the eye tracker.

To be able to perform a calibration the client must first enter the calibration state. The calibration state is an exclusive state and only one client can be the calibrating client at one time.

Some operations can only be performed during the calibration state, such as `AddCalibrationPoint()` [step 3], `RemoveCalibrationPoint()`, `ClearCalibration()` and `ComputeAndSetCalibration()`.

Other operations such as `SetCalibration()` or `GetCalibration()` work at any time. However, `SetCalibration()` can only be called by the client which is currently in the calibration state if a client is in the calibration state.

The calibration state is an exclusive state which can only be held by one client at a time. It is entered by calling the method `StartCalibration()` and is left by calling `StopCalibration()`. Whenever a client enters or leaves the calibration state the notifications `CalibrationStarted` and `CalibrationStopped` are sent to all clients except the calibrating client.

These notifications are mostly meant for UI display purposes, like graying out a “Calibrate” button etc. Since the communication with the eye tracker is asynchronous, best practice is to call `StartCalibration()` and check

whether it reports that another client is currently calibrating rather than caching the result of the calibration events .

Calibration Plot

If you have previous experiences of any of Tobii's eye tracking products it is very likely that you have seen the calibration plot which illustrates the calibration results. The calibration plot is a simple yet concise representation of a performed calibration and it usually looks something like what is shown in Figure 9. However, the presentation design can vary.

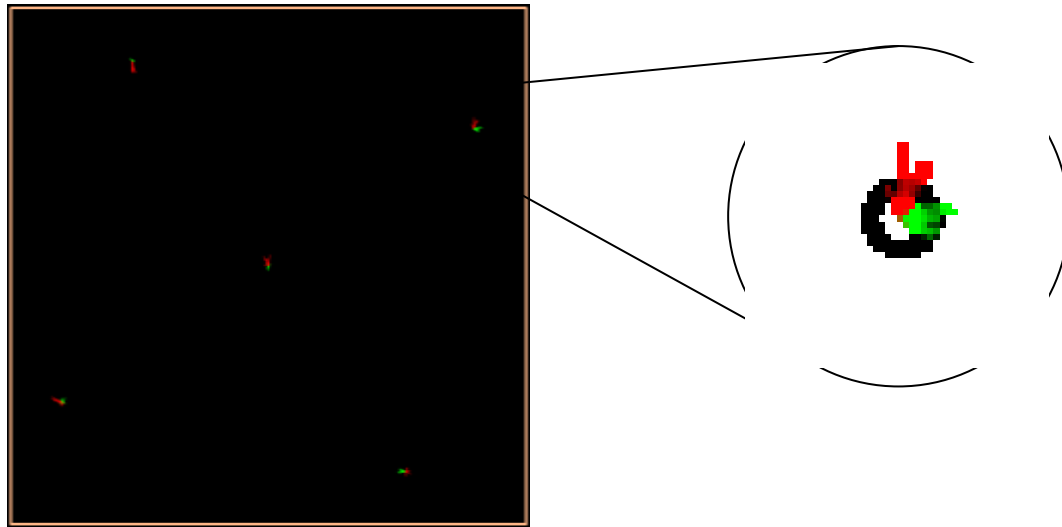


Figure 9: 5-point calibration plot with combined data for both the left and the right eye

The calibration plot shows the offset between the mapped gaze samples and the calibration points based on the best possible adaptation of the eye model to the collected values done by the eye tracker during calibration. In the above figure, the samples are drawn as small red and green lines representing the left and right eye respectively. The grey circles are the actual calibration points. In the .NET API, the calibration points are stored in the TrueX and TrueY members in the CalibrationPlotItem. In the Python API the calibration points are stored in the true_point member in the class OneCalibrationData.

Each sample also has a mapped point and validity per eye. In .NET these are MapLeftX, MapLeftY, ValidityLeft, MapRightX, MapRightY and ValidityRight. For python they are left.map_point.x, left.map_point.y and left.validity and analogously for the right eye. In the above graph each red and green line is a connection between the true point and a mapped point filtered by validity. The validity code can take the following values:

- -1 : eye not found
- 0 : found but not used
- 1 : used

Calibration Buffers

The eye tracker firmware uses two buffers to keep track of the calibration data.

- The Temporary Calibration Buffer. This buffer is used only during calibration. This is where data is added or removed.
- The Active Calibration Buffer. This buffer contains the data for the calibration that is currently set. This buffer is modified either by a call to `SetCalibration()`, which copies data from the client, or a successful call to `ComputeCalibration`, which computes calibration parameters based on the data in the temporary buffer and then copies the data into the active buffer.

Calibration Interface

C# Interface

Namespace	
Tobii.Eyetracking.Sdk	

Class/Interface	
IEyetracker	

Methods	Description
<code>void StartCalibration()</code>	Acquires the calibration state and clears the temporary calibration buffer.
<code>void StopCalibration()</code>	Releases the calibration state. This should always be done when the calibration is completed.
<code>void ClearCalibration()</code>	Clears the temporary calibration buffer.
<code>void AddCalibrationPoint(Point2D pt)</code>	Adds data to the temporary calibration buffer. The calibrating user is assumed to look at coordinate point, which is a normalized screen coordinate.
<code>void AddCalibrationPointAsync(Point2D pt, EventHandler<AsyncCompletedEventArgs<Empty>> responseHandler)</code>	Asynchronous version of the previous method. This can be useful to get a responsive UI, since the call may take some time.
<code>void RemoveCalibrationPoint(Point2D pt);</code>	Removes the data associated with a specific calibration point from the temporary calibration buffer.
<code>void ComputeCalibration()</code>	Uses the data in the temporary buffer and tries to compute calibration parameters. If the call is successful, the data is copied from the temporary buffer to the active buffer.
<code>void ComputeCalibrationAsync(EventHandler<AsyncCompletedEventArgs<Empty>> responseHandler)</code>	Asynchronous version of the previous method. This is useful as this operation can take a while on some eye trackers..
<code>Calibration GetCalibration()</code>	Gets the calibration data which is stored in the active calibration buffer. This data can be persisted to a file for later use.
<code>void SetCalibration(Calibration calibration)</code>	Sets the provided calibration, which means copying the data from the calibration into the active calibration

buffer.

Events	Description
event EventHandler<CalibrationStartedEventArgs> CalibrationStarted	This event is fired when another client acquires the calibration state.
event EventHandler<CalibrationStoppedEventArgs> CalibrationStopped	This event is fired when another client releases the calibration state.

Python Interface

Namespace
tobii.sdk.eyetracker

Class/Interface
Eyetracker

Methods	Description
def StartCalibration(self, callback = None, *args, **kwargs):	Acquires the calibration state and clears the temporary calibration buffer.
def StopCalibration(self, callback = None, *args, **kwargs):	Releases the calibration state. This should always be done when the calibration is finished.
def ClearCalibration(self, callback = None, *args, **kwargs):	Clears the temporary calibration buffer.
def AddCalibrationPoint(self, point, callback = None, *args, **kwargs):	Adds data to the temporary calibration buffer. The calibrating user is assumed to look at coordinate pt, which is a normalized screen coordinate.
def RemoveCalibrationPoint(self, point, callback = None, *args, **kwargs):	Removes the data associated with a specific calibration point from the temporary calibration buffer.
def ComputeCalibration(self, callback = None, *args, **kwargs):	Uses the data in the temporary buffer and tries to compute calibration parameters. If the call is successful, data is copied from the temporary buffer to the active buffer.
def GetCalibration(self, callback = None, *args, **kwargs):	Gets the calibration stored in the active calibration buffer. This data can be persisted to file for later use.
def SetCalibration(self, calibration, callback = None, *args, **kwargs):	Sets the provided calibration, which means copying the data from the calibration into the active calibration buffer.

Events	Description
<code>OnCalibrationStarted</code>	This event is fired when another client acquires the calibration state.
<code>OnCalibrationStopped</code>	This event is fired when another client releases the calibration state.

C++ Interface

Namespace
<code>Tobii::sdk::cpp::tracking</code>

Class/Interface
<code>Eyetracker</code>

Methods	Description
<code>void start_calibration()</code>	Acquires the calibration state and clears the temporary calibration buffer.
<code>void stop_calibration()</code>	Releases the calibration state. This should always be done when the calibration is finished!
<code>void clear_calibration()</code>	Clears the temporary calibration buffer.
<code>void add_calibration_point(const point_2d& point)</code>	Adds data to the temporary calibration buffer. The calibrating user is assumed to look at coordinate pt, which is a normalized screen coordinate.
<code>void add_calibration_point_async(const point_2d& point, const async_callback& completed_handler)</code>	Asynchronous version of the previous method. This can be useful to get a responsive UI, since the call may take some time.
<code>void remove_calibration_point(const point_2d& point)</code>	Removes the data associated with a specific calibration point from the temporary calibration buffer.
<code>void compute_calibration()</code>	Uses the data in the temporary buffer and tries to compute calibration parameters. If the call is successful, data is copied from the temporary buffer to the active buffer.
<code>void compute_calibration_async(const async_callback& completed_handler)</code>	Asynchronous version of the previous method, since this operation can take a while on some eye trackers.
<code>calibration get_calibration()</code>	Gets the calibration stored in the active calibration buffer. This data can be persisted to file for later use.
<code>void set_calibration(calibration& calibration)</code>	Sets the provided calibration, which means copying the data from the calibration into the active calibration buffer.
<code>connection add_calibration_started_listener(const empty_event::slot_type&</code>	Adds a listener to the calibration started event. This event is fired when another

<code>listener)</code>	client acquires the calibration state.
<code>connection add_calibration_stopped_listener(const empty_event::slot_type& listener)</code>	Adds a listener to the calibration stopped event. This event is fired when another client releases the calibration state.

Objective-C interface

Signature
<code>@class TET@EyeTracker</code>

Messages	Description
<code>-(void)startCalibration</code>	Acquires the calibration state and clears the temporary calibration buffer.
<code>-(void)stopCalibration</code>	Releases the calibration state. This should always be done when the calibration is finished!
<code>-(void)clearCalibration</code>	Clears the temporary calibration buffer.
<code>-(void)addCalibrationPoint:(NSPoint)p</code>	Adds data to the temporary calibration buffer. The calibrating user is assumed to look at coordinate pt, which is a normalized screen coordinate.
<code>-(void)removeCalibrationPoint:(NSPoint)p</code>	Removes the data associated with a specific calibration point from the temporary calibration buffer.
<code>-(void)computeCalibration</code>	Uses the data in the temporary buffer and tries to compute calibration parameters. If the call is successful, data is copied from the temporary buffer to the active buffer.
<code>@property (readwrite, assign) (TETCalibration*)calibration</code>	Gets the calibration stored in the active calibration buffer. This data can be persisted to file for later use. Sets the provided calibration, which means copying the data from the calibration into the active calibration buffer.

Related Delegate Messages	Description
<code>-(void)calibrationStarted</code>	This message is fired when another client acquires the calibration state.
<code>-(void)calibrationStopped</code>	This event is fired when another client releases the calibration state.

The Calibration class

The Calibration class is an in-memory representation of a personal calibration. It contains the raw data from the calibration as well as the calibration plot data. The raw data can be persisted to file and be used to recreate a calibration at a later point in time.

C# Interface

Namespace

Tobii.Eyetracking.Sdk

Signature

Public class Calibration

Constructors

Calibration(byte[] rawData)

Description

Creates a calibration object from binary data

Properties

List<CalibrationPlotItem> Plot {get;}

Description

Gets the calibration plot data contained in this calibration instance.

byte[] RawData {get; }

Get a binary representation of the calibration. This data can be persisted to file and used at a later point in time.

Python Interface

Namespace

tobii.sdk.converters

Signature

class Calibration(object):

Constructors

def __init__(self, blob):

Description

Creates a calibration object from binary data

Properties

**@property
def plot_data(self):**

Description

Gets the calibration plot data contained in this calibration instance.

self.rawData

Get a binary representation of the calibration. This data can be persisted to file and used at a later point in time.

C++ Interface

Namespace

tobii::sdk::cpp::tracking

Signature

class calibration

Constructors	Description
<code>calibration(const blob& data)</code>	Creates a calibration object from binary data

Methods	Description
<code>calibration_plot_item::vector_pointer get_plot_data()</code>	Gets the calibration plot data contained in this calibration instance.
<code>blob& get_raw_data()</code>	Get a binary representation of the calibration. This data can be persisted to file and used at a later point in time.

Objective-C interface

Signature
<code>@class TETCalibration</code>

Messages	Description
<code>-(NSData*)blob</code>	Get a binary representation of the calibration. This data can be persisted to file and used at a later point in time.
<code>-(NSArray*)plotItems</code>	Gets the calibration plot data contained in this calibration instance as an <code>NSArray*</code> containing <code>TETCalibrationPlotItem*</code>

Clock Functionality

Getting the Current Time

When writing an application that uses gaze data, it is often necessary to merge data from different sources. This in turn means that you have to compare the data using a common time base. With that in mind we have included some basic clock functionality in the SDK. The time functionality is exposed through the Clock class that is available for all supported languages and for all the different platforms. A Clock instance is a very simple object that can do two things:

- Provide the current time on the client computer. The time is given as a microsecond value stored in a 64 bit signed integer. The base time (t=0) is unspecified and can differ between different operating systems depending on the properties of the underlying hardware clock.
- Provide the clock resolution. This is a property of the clock and doesn't change over time. The resolution is also given as a microsecond value stored in a 64 bit signed integer.

The Clock class

C# Interface

Namespace
<code>Tobii.Eyetracking.Sdk.Time</code>

Signature
<code>public class Clock : IClock</code>

Constructors	Description
<code>Clock()</code>	Creates an instance of the default clock for the current system.

Methods	Description
<code>Int64 GetTime()</code>	Gets the current time as a microsecond value.
<code>Int64 GetResolution()</code>	Gets the clock resolution as a microsecond value.

Python Interface

Namespace
<code>tobii.sdk.time</code>

Signature
<code>class Clock(object):</code>

Constructors	Description
<code>def __init__(self):</code>	Creates an instance of the default clock.

Methods	Description
<code>def get_time(self):</code>	Gets the current time as a microsecond value.
<code>def get_resolution(self):</code>	Gets the clock resolution as a microsecond value.

C++ Interface

Namespace
<code>tobii::sdk::cpp::time</code>

Signature
<code>class clock</code>

Constructors	Description
<code>clock()</code>	Creates an instance of the default clock.

Methods	Description
<code>int64_t get_time()</code>	Gets the current time as a microsecond value.
<code>int64_t get_resolution()</code>	Gets the clock resolution as a microsecond value.

Objective-C interface

Signature
<code>@class TETClock</code>

Messages	Description
<code>-(int64_t)now</code>	Gets the current time as a microsecond value.
<code>-(int64_t)resolution</code>	Gets the clock resolution as a microsecond value.

Synchronizing clocks with the eye tracker

In order to correctly correlate stimulus events with gaze data, the high-resolution clock on the eye tracker must be synchronized with the high-resolution clock on the client computer. In previous versions of the SDK, this synchronization process has been completely hidden from the user. In this version, we have taken a more transparent approach. This allows for a better understanding of how the time synchronization works and also the possibility to handle errors that might occur in the synchronization.

The gaze data reported by the eye tracker is time stamped with a time from the high-resolution clock built into the eye tracker hardware. This clock is relative with no known start time. In the reminder of this document we will refer to this clock as the “eye tracker clock”.

Usually the time stamps from the clock on the client computer are of the same kind, i.e. relative and without known starting point. We refer to this clock as the “SDK clock”. The SDK clock can either be a built-in SDK clock which uses `QueryPerformanceCounter()` on Windows and `clock_gettime()` on POSIX. It is also possible to create a user-defined clock (for example an NTP clock) and use it with the SDK.

If the two clocks were to tick at exactly the same rate, it would be enough to compute the offset between them to get a decent synchronization. However, since the two clocks are based on computer hardware and no computer clocks are perfect, the two clocks will most likely drift in relation to each other. There are many reasons for this: One is that the clocks on a typical motherboard can have different tick rates due to imperfections in the clock crystals. Another reason is that the tick rate typically depends on external factors like temperature. This means that the tick rate of a clock can change over time. To account for this, the synchronization algorithm in the SDK always bases the synchronization on at least two measurements performed at different times. This gives us the possibility to not only estimate the clock offset, but also to estimate the drift of the two clocks.

The synchronization process is managed by a class in the SDK called the `SynchronizationManager`. This class is responsible for synchronizing a local clock (e.g. the SDK clock) with the high-resolution clock in the eye tracker. This class has two methods; `convert_from_local_to_remote()` and `convert_from_remote_to_local()` which can perform conversion from remote to local (i.e. client) timestamps and vice versa. These conversions are guaranteed to be performed offline, and so do not involve network traffic to the eye tracker.

The SynchronizationManager also has two events or callbacks: an error handler for asynchronous error reporting and a status changed handler which will be invoked when the synchronization status is updated or when a synchronization run has been performed.

The current synchronization status of a SynchronizationManager is described by a synchronization state flag which can have the following values:

- Unsynchronized:
 - The two clocks are completely unsynchronized
- Stabilizing:
 - Synchronization has started but only uses one sync point. This compensates for clock offset but not clock drift over time (see next section).
- Synchronized:
 - The two clocks are synchronized using a two point synchronization that takes into account both clock offset and clock drift (see next section).

There are two ways to get the synchronization status: either a query can be sent to the synchronization manager or a status changed event or received which indicates that synchronization has been performed. The synchronization status contains the following information:

- Synchronization state flag as previously described
- Points in Use:
 - A list of triplets (Local Midpoint, Remote timestamp and Roundtrip time) given in microseconds which describes the synchronization points currently in use
- Error Approximation:
 - An approximation of the current synchronization error which is defined as the maximum roundtrip time in “Point in Use” divided by two.

Synchronization Process

Currently the synchronization uses Cristian’s Algorithm⁵. At fixed, growing intervals the synchronization manager will ask the eye tracker for its time and from that drift and offset will be computed so that timestamps can be converted between the SDK clock and the eye tracker clock without requiring any network traffic.

⁵ http://en.wikipedia.org/wiki/Cristian's_algorithm

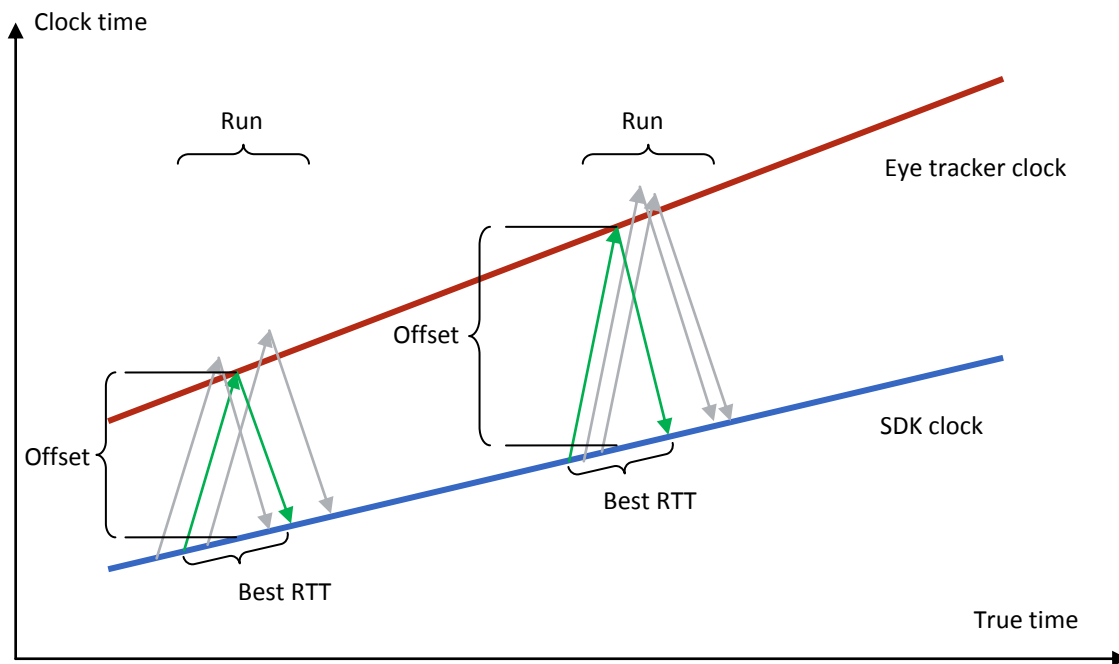


Figure 10: Synchronization procedure

The synchronization manager will at regular intervals perform a synchronization run. From each run the synchronization point with the shortest roundtrip gets selected. A synchronization point consists of two local timestamps and one remote timestamp like this

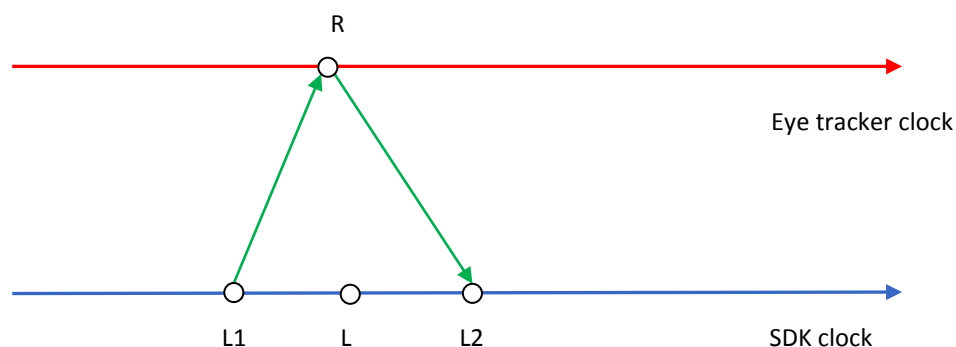


Figure 11: Synchronization Sample

From this data we can now estimate the relationship between the two clocks by assuming that $L=R$, where L is computed as the mean value of $L1$ and $L2$. The roundtrip time ($RTT=L2-L1$) gives an estimate of the synchronization error in such a way that the maximum synchronization error is bounded by half the roundtrip time ($RTT/2$). By using two synchronization points we calculate both the offset and drift between the two clocks by using the formulas as seen below

$$drift = \frac{r_2 - r_1}{l_2 - l_1}$$

$$offset = r_1 - drift \times l_1$$

Where r is the timestamp from the eye tracker clock and l is the midpoint of local start and end points (from the SDK clock). The indices indicate sync point 1 and 2.

Conversion from the SDK clock to the eye tracker clock and vice versa can then be performed by:

$$t_{eyetracker} = drift \times t_{sdk} + offset$$

$$t_{sdk} = \frac{t_{eyetracker} - offset}{drift}$$

The conversion is performed by the function `convert_from_local_to_remote()` and `convert_from_remote_to_local()` in the synchronization manager class.

Synchronization Scenarios

The synchronization setups can be grouped into a couple of scenarios where the synchronization subsystem is used in different ways. By performing synchronization it is possible to get comparable timestamps in gaze data and local events such as stimuli and user interaction.

One stimuli computer and one eye tracker

The simplest case is when one stimuli computer and one eye tracker is used:

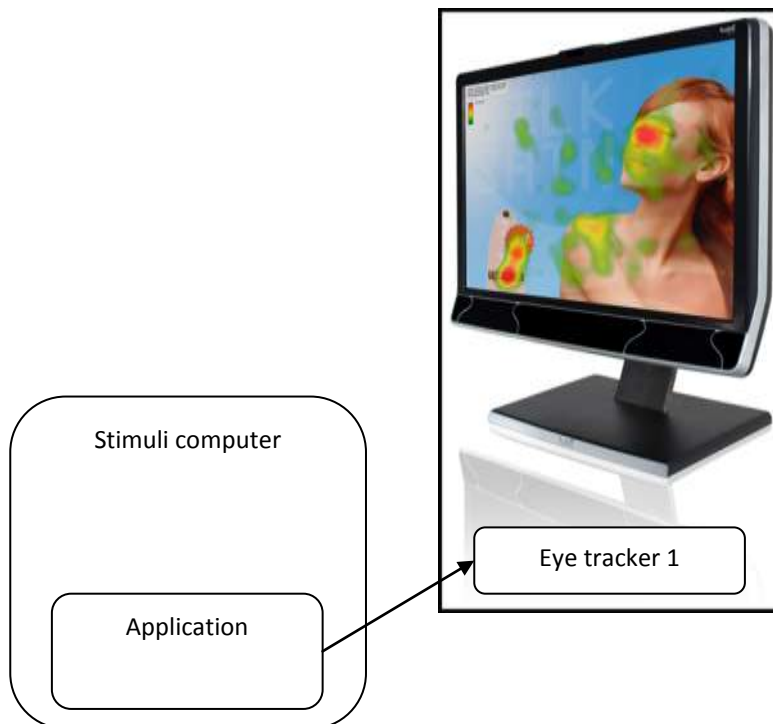


Figure 12: Standard Synchronization Scenario

In this setup we have only two clocks involved: the SDK clock on the stimuli computer and the eye tracker's clock.

By using a synchronization manager we can synchronize the SDK clock with the eye tracker's clock. We could choose either the local SDK clock or the remote clock as our base timeline. In this case it doesn't matter which one we choose as either choice will give the same result.

One stimuli computer and several eye trackers

A more complicated case is when we have one stimuli computer and several eye trackers:

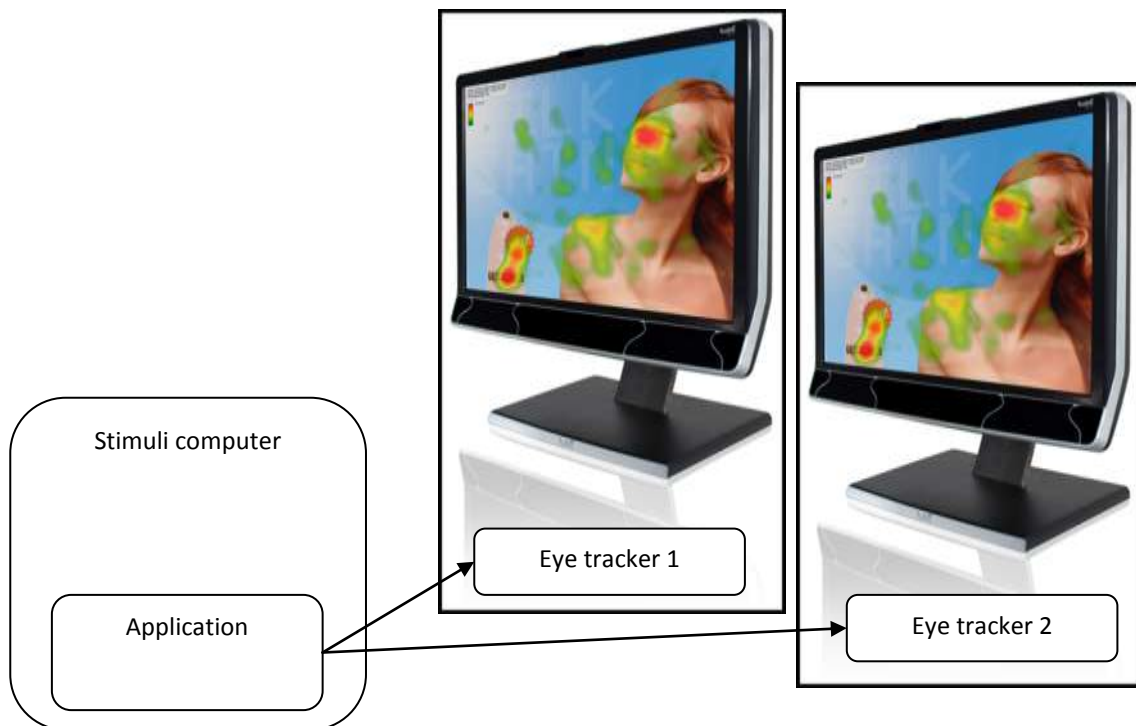


Figure 13: Synchronization Scenario with several eye trackers

In this setup we have several clocks involved: first the SDK clock on the stimuli computer and then each eye tracker has its own internal clock. Hence, in this situation there are three clocks that need to be synchronized.

By using two synchronization managers we can synchronize the SDK clock with both of the eye trackers' clocks. As we have more than one remote clock, it makes sense to use the SDK clock as the time source of choice and convert all timestamps from both of the eye trackers to the SDK clock. Technically we could use any of the clocks as a common base. For example we could just as well convert timestamps originating from eye tracker 1 to the time base of eye tracker 2 by first adjusting them to local time of the stimuli computer. However, this would cause a loss of accuracy as each conversion is affected by the current synchronization error on that synchronization.

We will however require one synchronization manager to handle each eye tracker in the system.

Two (or more) stimuli computers and one eye tracker

A slightly more advanced case involves several stimuli computers sharing the same eye tracker:

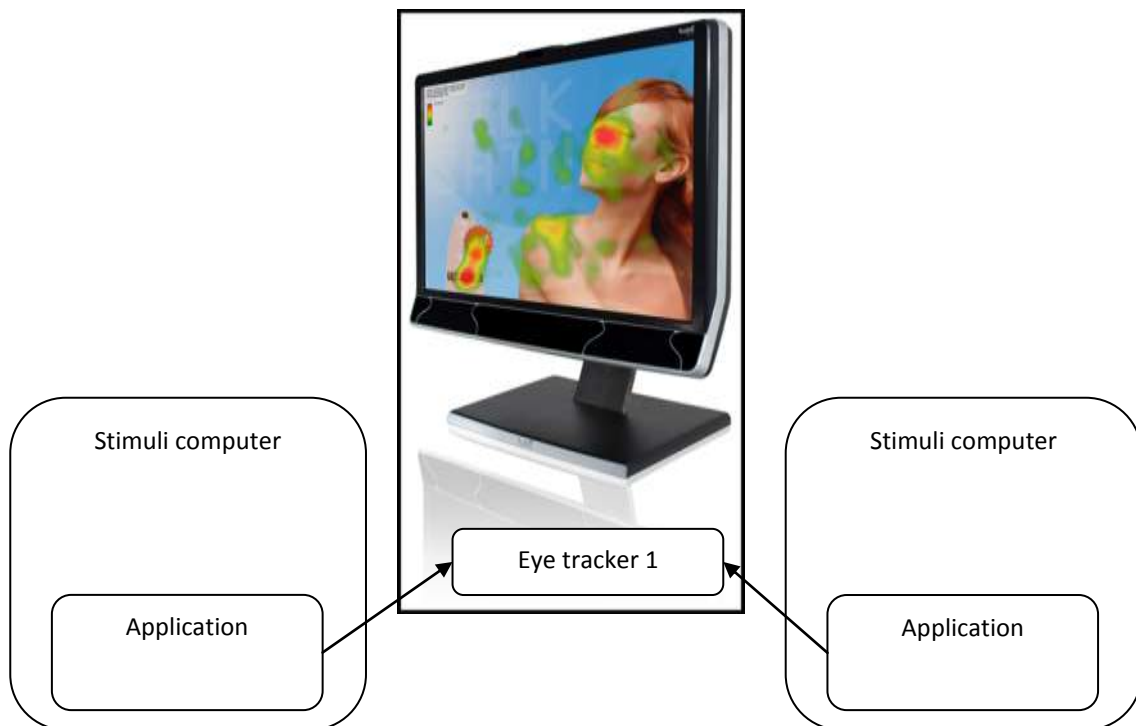


Figure 14: Synchronization Scenario with one eye tracker and several applications

Also in this setup we have several clocks involved: we have one SDK clock per stimuli computer and one eye tracker clock.

Each stimuli computer can synchronize with the eye tracker by means of a synchronization manager. However, because the synchronization manager cannot synchronize two SDK clocks we must use a different approach.

Each application must use a synchronization manager to synchronize with the eye tracker. In order to get comparable timestamps in both applications, they must then convert their local timestamps (stimuli events) to the eye tracker's clock. This means that the eye tracker's internal clock will act as the common time base for both stimuli computers.

Several setups of stimuli computers and eye trackers

A really advanced case involves several stimuli computers and one or more eye trackers:

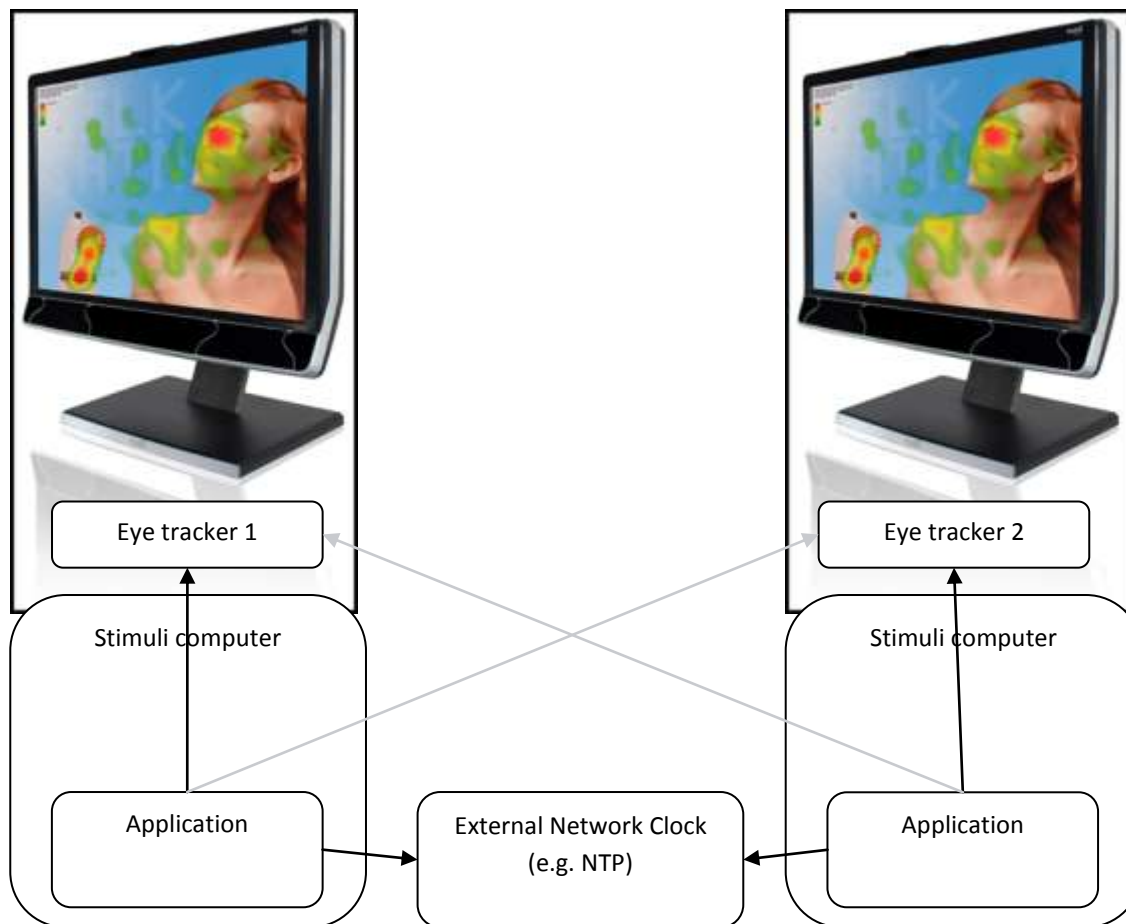


Figure 15: Synchronization Scenario with multiple eye trackers and applications and external clock source

The main issue with this setup is that there is no part of it that can be used as a common time base for all the other components involved.

There are two possible solutions: all applications on the stimuli computers synchronize with all eye trackers in the system where one common eye tracker is used as the common time source (this solution uses the grey arrows in Figure 15).

The SyncManager class

C# Interface

Namespace	
<code>Tobii.Eyetracking.Sdk.Time</code>	
Signature	
<code>class SyncManager : ISyncManager, IDisposable</code>	
Constructors	Description

<code>SyncManager(Clock clock, EyetrackerInfo eyetrackerInfo)</code>
<code>SyncManager(Clock clock, FactoryInfo factoryInfo)</code>
<code>SyncManager(Clock clock, EyetrackerInfo eyetrackerInfo, EventThreadingOptions threadingOptions)</code>
<code>SyncManager(Clock clock, FactoryInfo factoryInfo, EventThreadingOptions threadingOptions)</code>

Properties	Description
<code>SyncState SyncState { get; }</code>	Gets the current synchronization state.

Methods	Description
<code>Int64 RemoteToLocal(Int64 remoteTime)</code>	Converts from remote (eye tracker) time to local time. The time unit is always microseconds.
<code>Int64 LocalToRemote(Int64 localTime)</code>	Converts from local to remote (eye tracker) time. The time unit is always microseconds.

Events	Description
<code>event EventHandler<SyncStateChangedEventArgs> SyncStateChanged;</code>	This event is fired when the synchronization state changes. The event argument contains the updated sync state information similar to what you get from the SyncState property.

Python Interface

Namespace
<code>tobii.sdk.time.sync</code>

Signature
<code>class SyncManager(object):</code>

Constructors	Description
<code>def __init__(self, clock, eyetracker_info, mainloop, error_handler=None, status_handler=None):</code>	

Properties	Description
------------	-------------

<code>@property def sync_state(self):</code>	Gets the current synchronization state
--	--

Methods	Description
<code>def convert_from_remote_to_local(self, remote_usecs):</code>	Converts from remote (eye tracker) time to local time. The time unit is always microseconds.
<code>def convert_from_local_to_remote(self, local_usecs):</code>	Converts from local to remote (eye tracker) time. The time unit is always microseconds.

C++ Interface

Namespace
<code>tobii::sdk::cpp::time</code>

Signature
<code>class sync_manager</code>

Constructors	Description
<code>sync_manager(const mainloop& mainloop, const clock& clock, const discovery::factory_info& factory_info)</code>	

Methods	Description
<code>int64_t remote_to_local(int64_t remote_time)</code>	Converts from remote (eye tracker) time to local time. The time unit is always microseconds.
<code>int64_t local_to_remote(int64_t local_time)</code>	Converts from local to remote (eye tracker) time. The time unit is always microseconds.
<code>sync_state::pointer get_sync_state()</code>	Gets the current synchronization state
<code>connection add_sync_state_changed_listener(const sync_state_changed_event::slot_type& listener)</code>	Adds a listener to the sync state changed event, which is fired when the synchronization state is updated. The event argument contains the updated sync state information similar to what you get from the <code>get_sync_state()</code> method.

Objective-C interface

Signature
<code>@class TETSyncManager</code>

Messages	Description
<code>@property (readwrite, assign) id<TETSyncManagerDelegate> delegate</code>	Gets or sets the delegate messages are sent to when the sync manager changes state or encounter an error.
<code>-(int64_t)remoteToLocal:(int64_t)remoteTime</code>	Converts from remote (eye tracker) time to local time. The time unit is always microseconds.
<code>-(int64_t)localToRemote:(int64_t)localTime</code>	Converts from local to remote (eye tracker) time. The time unit is always microseconds.
<code>-(TETSyncState*) syncState</code>	Gets the current synchronization state

Getting and Setting Eye Tracker Properties

All Tobii Eye Trackers have a set of properties that can be queried for and sometimes also set. This functionality is provided by the Eyetracker class/interface and is described in the following sections.

Basic Eye Tracker Info

The data contained in the EyetrackerInfo class that you get from the browsing is also available through the EyeTracker interface.

C# Interface

Namespace
<code>Tobii.Eyetracking.Sdk</code>

Class/Interface
<code>IEyetracker</code>

Methods	Description
<code>UnitInfo GetUnitInfo()</code>	Gets a UnitInfo instance containing eye tracker information through the eyetracker interface
<code>string GetUnitName()</code>	Gets the eye tracker's name.
<code>void SetUnitName(string name)</code>	Sets the eye tracker's name.

Python Interface

Namespace
<code>tobii.sdk.eyetracker</code>

Class/Interface	
Eyetracker	
Methods	Description
<code>def GetUnitInfo(self, callback = None, *args, **kwargs):</code>	Gets a UnitInfo instance containing eye tracker information through the eyetracker interface.
<code>def GetUnitName(self, callback = None, *args, **kwargs):</code>	Gets the eye tracker's name.
<code>def SetUnitName(self, name, callback = None, *args, **kwargs):</code>	Sets the eye tracker's name.

C++ Interface

Namespace	
<code>tobii::sdk::cpp::tracking</code>	
Class/Interface	
Eyetracker	
Methods	Description
<code>unit_info::pointer get_unit_info()</code>	Gets a UnitInfo instance containing eye tracker information through the eyetracker interface.
<code>std::string get_unit_name()</code>	Gets the eye tracker's name.
<code>void set_unit_name(const std::string& name)</code>	Sets the eye tracker's name.

Objective-C interface

Signature	
<code>@class TETEyeTracker</code>	
Messages	Description
<code>-(TETUnitInfo*) unitInfo</code>	Gets a UnitInfo instance containing eye tracker information.
<code>@property (readwrite, assign) NSString* name</code>	Gets or sets the eye tracker's name.

Getting and Setting the Eye tracker's Frame rate

All Tobii Eye Trackers supports querying for the current sampling frequency or *frame rate*. On some models, e.g., T120 and TX300, it is also possible to set the frame rate.

C# Interface

Namespace	
Tobii.Eyetracking.Sdk	

Class/Interface	
IEyetracker	

Methods	Description
<code>float GetFramerate()</code>	Gets the current framerate. A value of, e.g., 60.0 indicates that the current framerate is 60 Hz.
<code>IList<float> EnumerateFramerates()</code>	Gets a list of possible framerates for this eye tracker.
<code>void SetFramerate(float framerate)</code>	Sets the current framerate. The framerate value must be one of the values returned from EnumerateFramerates or this method will fail.

Events	Description
<code>event EventHandler<FramerateChangedEventArgs> FramerateChanged</code>	Occurs when <i>another</i> client changes the framerate.

Python Interface

Namespace	
tobii.sdk.eyetracker	

Class/Interface	
Eyetracker	

Methods	Description
<code>def GetFramerate():</code>	Gets the current framerate. A value of, e.g., 60.0 indicates that the current framerate is 60 Hz.
<code>def EnumerateFramerates(self, callback, *args, **kwargs)</code>	Gets a list of possible framerates for this eye tracker.
<code>def SetFramerate(self, framerate, callback, *args, **kwargs)</code>	Sets the current framerate. The framerate value must be one of the values returned from EnumerateFramerates or this method will fail.

Events	Description
<code>OnFramerateChanged</code>	Occurs when <i>another</i> client changes the framerate.

C++ Interface

Namespace
<code>tobii::sdk::cpp::tracking</code>

Class/Interface
<code>Eyetracker</code>

Methods	Description
<code>float get_framerate()</code>	Gets the current framerate. A value of, e.g., 60.0 indicates that the current framerate is 60 Hz.
<code>std::vector<float> enumerate_framerates()</code>	Gets a list of possible framerates for this eye tracker.
<code>void set_framerate(float framerate)</code>	Sets the current framerate. The framerate value must be one of the values returned from <code>EnumerateFramerates</code> or this method will fail.
<code>connection add_framerate_changed_listener(const framerate_changed_event::slot_type& listener)</code>	Adds a listener for the framerate changed event. This event is fired when <i>another</i> client changes the framerate.

Objective-C interface

Signature
<code>@class TETEyetracker</code>

Messages	Description
<code>@property (readwrite, assign) float framerate</code>	Gets or sets the framerate for this eye tracker. Only values returned from <code>availableFramerates</code> can be set as framerate.
<code>-(NSArray*) availableFramerates</code>	Gets a list of possible framerates for this eye tracker. Each element is a float.

Related Delegate Messages	Description
<code>-(void) framerateChanged</code>	This message is sent to the delegate when <i>another</i> client changes the framerate.

Querying the Eye Tracker for its Head Movement Box

The Tobii SDK 3.0 contains an interface for querying the eye tracker for the current head movement box. The head movement box is defined as the volume in front of the eye tracker in which tracking is possible. This information can for example be used to give the user feedback on her position in front of the eye tracker. Note

that the head movement box is different between different eye tracker models and can change depending on frame rate.

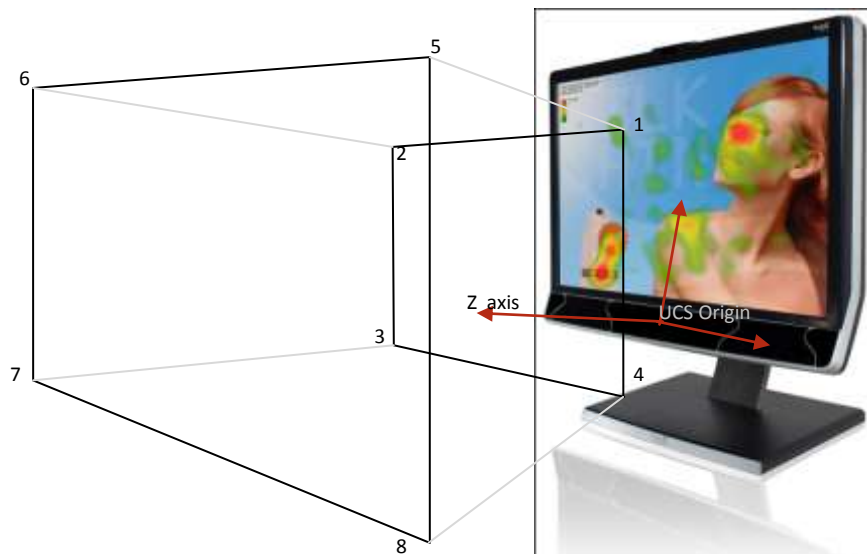


Figure 16: Schematic head movement box

As can be seen from the schematic above (i.e. Figure 16), the head movement box is a frustum described by eight points. The points are reported in the User Coordinate System in millimeters. The coordinate system has its origin where the marker “UCS Origin” is in the figure.

The head movement box as reported by the firmware is only an approximation of the actual head movement box and it is only intended to be used for demonstration purposes in order to guide the user to the correct location in front of the eye tracker.

The points in the far end of the box in relation to the eye tracker (5-8) and in the near plane (1-4) have the same Z-coordinates internally. This provides the possibility to compute a relative distance between these planes taking any Z in the near and far plane and compute the relative Z coordinate as seen below:

$$\text{Relative distance} = \frac{Z_{eye} - Z_1}{Z_5 - Z_1}$$

C# Interface

Namespace

Tobii.Eyetracking.Sdk

Class/Interface

IEyetracker

Methods

HeadMovementBox GetHeadMovementBox ()

Description

Gets the current head movement box. The HeadMovementBox class is just a POD that contains eight 3D points as described above.

Events	Description
event EventHandler<HeadMovementBoxChangedEventArgs> HeadMovementBoxChanged	Occurs when the head movement box changes. This happens on some systems, e. g., on the T120, when you change the framerate.

Python Interface

Namespace
tobii.sdk.eyetracker

Class/Interface
Eyetracker

Methods	Description
def GetHeadMovementBox(self, callback, *args, **kwargs)	Gets the current head movement box. Returns a HeadMovementBox class that is just a POD containing eight 3D points as described above.

Events	Description
OnHeadMovementBoxChanged	Occurs when the head movement box changes. This happens on some systems, e. g., on the T120, when you change the framerate.

C++ Interface

Namespace
tobii::sdk::cpp::tracking

Class/Interface
eyetracker

Methods	Description
head_movement_box::pointer get_head_movement_box()	Gets the current head movement box. Returns a shared pointer to a head_movement_box structure containing eight 3D points as described above.
connection add_headbox_changed_listener(const empty_event::slot_type& listener)	Adds a listener for the headbox changed event. This happens on some systems, e. g., on the T120, when you change the framerate.

Objective-C interface

Signature	
<code>@class TETEyeTracker</code>	

Messages	Description
<code>- (TETHeadBox*) headBox;</code>	Gets the current head movement box. Returns a shared pointer to a head_movement_box structure containing eight 3D points as described above.

Related Delegate Messages	Description
<code>- (void) headBoxChanged</code>	Message sent to the delegate when the head movement box changes. This happens on some systems, e. g., on the T120, when you change the framerate.

Configuring the Screen (only X-Units)

When using the Tobii X-series Eye Trackers it is necessary to configure the eye tracker- and calibration area geometry. This has previously only been possible to do through the X120 configuration tool found in the Eyetracker Browser application. This application is bundled with Tobii Studio as well as the previous versions of the SDK. The new version of the Tobii SDK contains functionality to set and get the so called x-configuration parameters without having to use the X120 configuration tool.

Before starting to use an X-series eye tracker, it is necessary to inform the eye tracker about the relative position of the calibration area in relation to the eye tracker. This area usually corresponds to some kind of monitor or projector screen, but could be any planar surface. The calibration area is described by three points in space, the upper left, lower left and upper right corner of the plane. The three points **must** be described in the so called user coordinate system (or *UCS* for short) illustrated by red vectors in Figure 17. This is a millimeter based coordinate system with its origin in the middle of the front filter surface. The x-axis points to the user's right, the y-axis points in the upward direction and the z-axis points towards the user. Note that this coordinate system is eye tracker fixed, which means that tilting or moving the eye tracker requires the user to provide new coordinates for the calibration area.

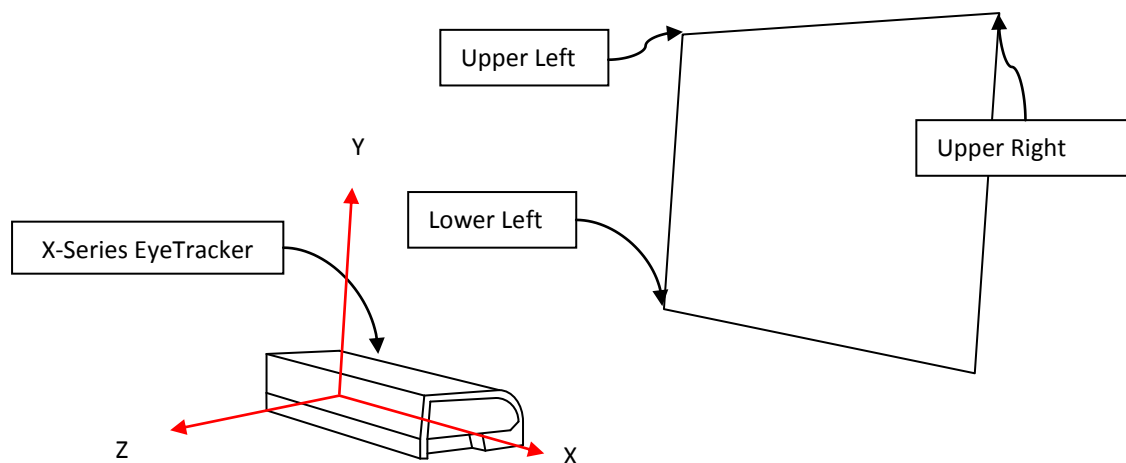


Figure 17: Screen Configuration Schematic

Note: It is important that the provided points correspond to the two sides of a rectangle. If the correct coordinates are not provided to the eye tracker, this will result in offsets in the gaze data.

It is also possible to query the eye tracker for the current calibration area coordinates. This can be done for T-series eye trackers as well as for X-series eye trackers.

API for Configuring the Screen

C# Interface

Namespace	
<code>Tobii.Eyetracking.Sdk</code>	
Class/Interface	
<code>IEyetracker</code>	
Methods	Description
<code>void SetXConfiguration(XConfiguration configuration);</code>	Sets the x configuration parameters. This method will only work on x-units, e.g., the X120.
<code>XConfiguration GetXConfiguration();</code>	Gets the X configuration parameters.
Events	Description
<code>event EventHandler<XConfigurationChangedEventArgs> XConfigurationChanged</code>	This event is fired when the X configuration changes.

Python Interface

Namespace	
<code>tobii.sdk.eyetracker</code>	
Class/Interface	
<code>Eyetracker</code>	
Methods	Description
<code>def SetXConfiguration(self, UpperLeft, UpperRight, LowerLeft, callback = None, *args, **kwargs):</code>	Sets the X configuration parameters. This method will only work on x-units, e.g., the X120.
<code>def GetXConfiguration(self, callback = None, *args, **kwargs):</code>	Gets the X configuration parameters.
Events	Description
<code>OnXConfigurationChanged</code>	This event is fired when the X configuration changes.

C++ Interface

Namespace	
<code>tobii::sdk::cpp::tracking</code>	
Class/Interface	
<code>eyetracker</code>	
Methods	Description
<code>void set_x_configuration(const x_configuration& configuration)</code>	Sets the X configuration parameters. This method will only work on x-units, e.g., the X120.
<code>x_configuration::pointer get_x_configuration()</code>	Gets the X configuration parameters.
<code>connection add_xconfig_changed_listener(const x_configuration_changed_event::slot_type& listener)</code>	Adds a listener for the X configuration changed event. This event is fired when the X configuration changes.

Objective-C Interface

Signature

```
@class TETEyeTracker
```

Messages	Description
<pre>@property (readwrite, assign) TETXConfiguration* xConfiguration</pre>	Sets or gets the X configuration parameters. This method will only work on x-units, e.g., the X120.

Related Delegate Messages	Description
<pre>-(void)xconfigChanged</pre>	This message is sent to the delegate when the X configuration changes.

Extended Functionality

This section contains a list of methods that exist in the eye tracker interface that makes it possible to activate special functionality in the eye tracker, such as debug functionality. As a developer you don't normally have to care about these methods but they are listed for the sake of completeness.

C# Interface

Namespace
<code>Tobii.Eyetracking.Sdk</code>

Class/Interface
<code>IEyetracker</code>

Methods	Description
<code>IList<Extension> GetAvailableExtensions()</code>	Gets a list of all available extensions.
<code>IList<Extension> GetEnabledExtensions()</code>	Gets a list of all enabled extensions.
<code>void EnableExtension(Int32 extensionId)</code>	Enables a specific extension.
<pre>AuthorizeChallenge GetAuthorizeChallenge(Int32 realmId, IList<Int32> algorithms)</pre>	Gets an authorization challenge to unlock custom functionality.
<pre>void ValidateChallengeResponse(Int32 realmId, Int32 algorithm, byte[] responseData)</pre>	Validates and unlocks custom functionality.
<code>void DumpImages(Int32 count, Int32 frequency)</code>	Method used to diagnose hardware problems.
<code>byte[] GetDiagnosticReport(Int32 includeImages)</code>	Method used to diagnose hardware problems.
<code>PayperuseInfo GetPayperuseInfo()</code>	Gets information whether this is a pay per use eye tracker or not.

Python Interface

Namespace	
<code>tobii.sdk.eyetracker</code>	
Class/Interface	
<code>Eyetracker</code>	
Methods	Description
<code>def GetAvailableExtensions(self, callback = None, *args, **kwargs):</code>	Gets a list of all available extensions.
<code>def GetEnabledExtensions(self, callback = None, *args, **kwargs):</code>	Gets a list of all enabled extensions.
<code>def EnableExtension(self, extensionId, callback = None, *args, **kwargs):</code>	Enables a specific extension.
<code>def GetAuthorizeChallenge(self, realmId, algorithms, callback = None, *args, **kwargs):</code>	Gets an authorization challenge to unlock custom functionality.
<code>def ValidateChallengeResponse(self, realmId, algorithm, responseData, callback = None, *args, **kwargs):</code>	Validates and unlocks custom functionality.
<code>def DumpImages(self, count, frequency, callback = None, *args, **kwargs):</code>	Method used to diagnose hardware problems.
<code>def GetDiagnosticReport(self, include_images, callback = None, *args, **kwargs):</code>	Method used to diagnose hardware problems.
<code>def GetPayperuseInfo(self, callback = None, *args, **kwargs):</code>	Gets information whether this is a pay per use eye tracker or not.

C++ Interface

Namespace	
<code>tobii::sdk::cpp::tracking</code>	
Class/Interface	
<code>eyetracker</code>	
Methods	Description

<code>extension::vector_pointer get_available_extensions()</code>	Gets a list of all available extensions.
<code>extension::vector_pointer get_enabled_extensions()</code>	Gets a list of all enabled extensions.
<code>void enable_extension(uint32_t extension_id)</code>	Enables a specific extension.
<code>authorize_challenge get_authorize_challenge(uint32_t realm_id, const std::vector<uint32_t> algorithms)</code>	Gets an authorization challenge to unlock custom functionality.
<code>void validate_challenge_response(uint32_t realm_id, uint32_t algorithm, blob& response_data)</code>	Validates and unlocks custom functionality.
<code>void dump_images(uint32_t count, uint32_t frequency)</code>	Method used to diagnose hardware problems.
<code>blob get_diagnostics_report(uint32_t include_images)</code>	Method used to diagnose hardware problems.
<code>payperuse_info get_payperuse_info()</code>	Gets information whether this is a pay per use eye tracker or not.

Objective-C Interface

Signature
<code>@class TETEyeTracker</code>

Messages	Description
<code>-(NSArray*) availableExtensions</code>	Gets a list of all available extensions. Each element is a <code>TETExtension*</code> .
<code>-(NSArray*) enabledExtensions</code>	Gets a list of all enabled extensions. Each element is a <code>TETExtension*</code> .
<code>-(void) enableExtension: (TETExtension*) extension</code>	Enables a specific extension.
<code>-(TETAuthorizeChallenge*) authorizeRealWithId: (uint32_t) realmId knownAlgorithms: (NSArray*) algorithms;</code>	Gets an authorization challenge to unlock custom functionality.
<code>(NSError*) validateChallengeWithRealmId: (uint32_t) realmId usingAlgorithm: (uint32_t) algorithm response: (NSData*) responseData</code>	Validates and unlocks custom functionality.
<code>-(void) dumpImages: (uint32_t) count withFrequency: (uint32_t) frequency</code>	Method used to diagnose hardware problems.
<code>-(NSData*) diagnosticsReportIncludeImages: (BOOL) includeImages</code>	Method used to diagnose hardware problems.
<code>payperuse_info get_payperuse_info()</code>	Gets information whether this is a pay per use eye tracker or not.

Firmware Upgrade

The Tobii SDK 3.0 contains functionality for upgrading the eye tracker firmware. This has previously only been possible to do from the Eyetracker Browser application that is shipped as a part of the old SDK as well as with the Tobii Studio application. Doing a firmware upgrade is quite straightforward. A Tobii firmware package file, usually with file extension `.tobiipkg`, is uploaded to a selected eye tracker and automatically installed if certain compatibility requirements are met. Since the upgrade procedure may take some time, progress is continuously reported back to the upgrading application until the upgrade procedure is completed.

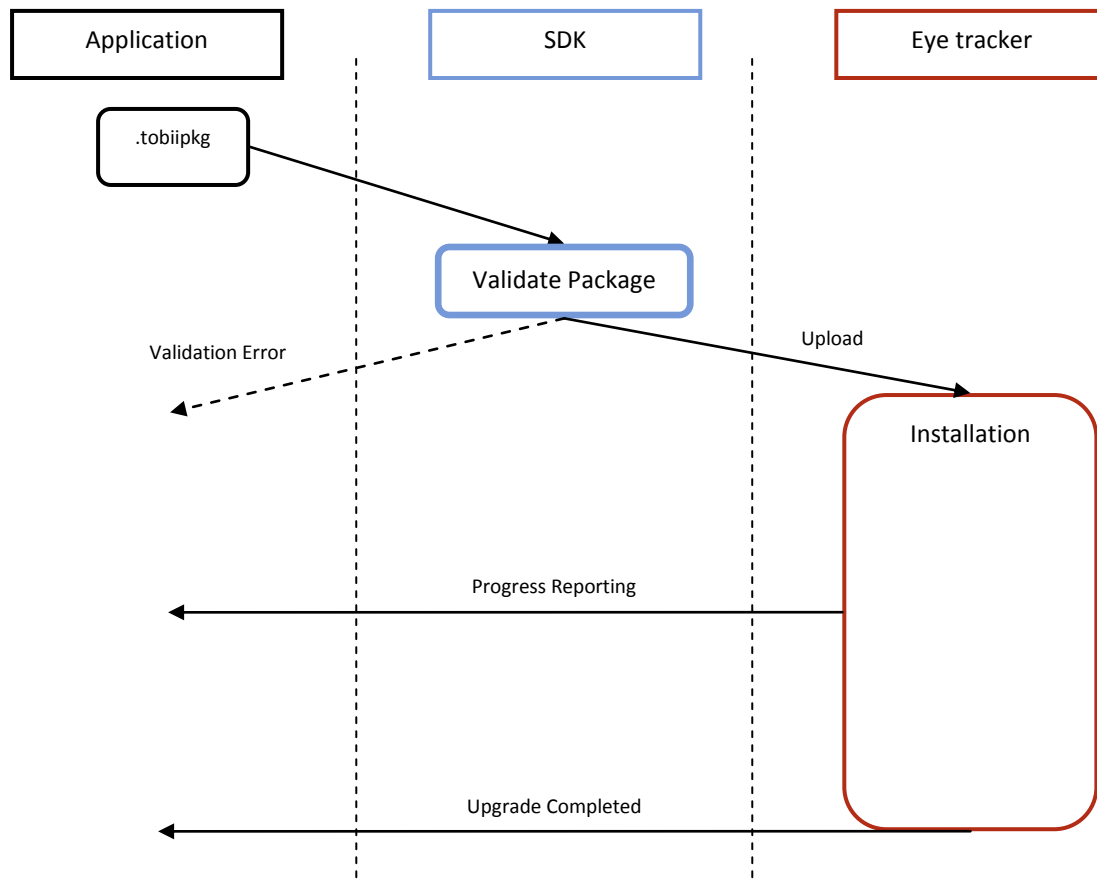


Figure 18: Upgrade Procedure

Upgrade Functions/Methods

C# Interface

Namespace
<code>Tobii.Eyetracking.Sdk.Upgrade</code>
Signature
<code>public static class UpgradeManager</code>

Methods	Description
<pre>static bool UpgradePackageIsCompatibleWithDevice(string filePath, EyetrackerInfo unitToUpgrade, out Int32 errorCode)</pre>	Checks if an upgrade package is compatible with a specific eye tracker.
<pre>static void BeginUpgrade(string filePath, EyetrackerInfo unitToUpgrade, ProgressReporter progressReporter)</pre>	<p>Starts the upgrade. The process is asynchronous, which means that this call will return immediately.</p> <p>Parameters:</p> <p>filePath: path to a .tobipkg upgrade package.</p> <p>unitToUpgrade: the eye tracker to upgrade</p> <p>progressReporter: a ProgressReporter instance that will report when the upgrade procedure fails or completes.</p>

Python Interface

Namespace
tobii.sdk.upgrade

Functions	Description
<pre>def package_is_compatible_with_device(mainloop, package_path, device_info):</pre>	Checks if an upgrade package is compatible with a specific eye tracker.
<pre>def begin_upgrade(mainloop, package_path, device_info, completed_handler, progress_handler, cancel_handler):</pre>	Starts the upgrade. The process is asynchronous, which means that this call will return immediately.

C++ Interface

Namespace
tobii::sdk::cpp::upgrade

Functions	Description
<pre>bool upgrade_package_is_compatible_with_device(const mainloop& mainloop, const upgrade_package& package, const eyetracker_info& info, uint32_t& error_code)</pre>	Checks if an upgrade package is compatible with a specific eye tracker.
<pre>void begin_upgrade(const mainloop& mainloop, const upgrade_package& package, const eyetracker_info& info, const progress_reporter& reporter)</pre>	Starts the upgrade. The process is asynchronous, which means that this call will return immediately.

Parameters:

mainloop: a mainloop instance.**package:** the upgrade package to use**info:** the eye tracker to upgrade**reporter:** a ProgressReporter instance that will report when the upgrade procedure fails or completes.**Objective-C Interface**

The objective-C interface provides a subclass of `NSOperation` – `TETUpgradeOperation` that can easily be used together with an `NSOperationQueue`. The progress property supports Key-Value Observation (KVO). This allows for easy implementation of progress bars. The `TETUpgradeOperation` can also be used without an `NSOperationQueue` by sending it a `start` message. See the `NSOperation` documentation for more information.

Signature

```
@class TETUpgradeOperation
```

Messages	Description
<code>-(id)initWithEyeTracker:(TETEyeTrackerInfo*)info andUpgradePackage:(TETUpgradePackage*)package;</code>	Initiates this <code>TETUpgradeOperation</code> with the given eye tracker and upgrade package.
<code>@property (readwrite, assign) double progress</code>	Gets the progress of the current upgrade step. Supports KVO.
<code>@property (readwrite, assign) NSUInteger numberOfSteps</code>	Gets the number of steps in the upgrade process.
<code>@property (readwrite, assign) NSUInteger step</code>	Gets the current step in the upgrade process. Supports KVO.
<code>@property (readwrite, retain) NSError* error</code>	Gets the error, if any, caused by this upgrade operation.

Signature

```
@class TETUpgradePackage
```

Messages	Description
<code>-(id)initWithBlob:(NSData*)blob</code>	Creates an upgrade package from the information in the <code>NSData*</code>
<code>-(id)initWithFileHandle:(NSFileHandle*)handle</code>	Creates an upgrade package from the contents in the given file handle.
<code>-(id)initWithPath:(NSString*)path</code>	Creates an upgrade package from the contents in the file given by <code>path</code> .
<code>-(BOOL)isCompatibleWithEyeTracker:(TETEyeTrackerInfo*)info</code>	Checks if an upgrade package is compatible with a specific eye tracker.
<code>@property (readwrite, retain) NSError* error</code>	Gets the error, if any, caused by this upgrade operation.

Troubleshooting

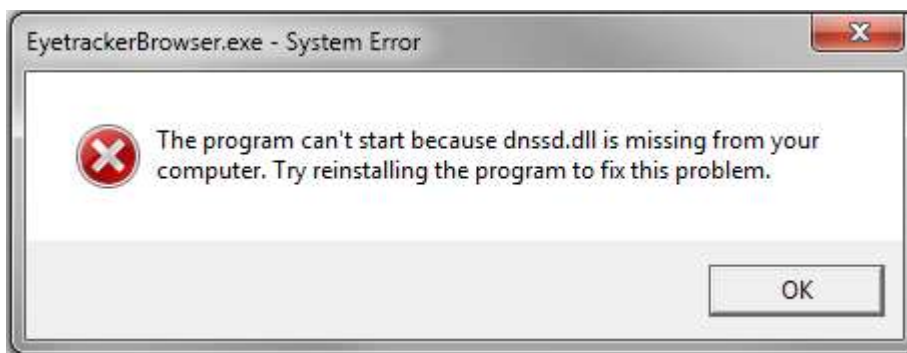
Logging

The TobiiSDK can print information about its inner workings to a log file. To enable this feature set the following environment variables before launching the process using the Tobii SDK.

Name	Value
TOBII_SDK_LOG_LEVEL	Sets the log level. Can be any of log4cpp levels: "ERROR, TRACE,"
TOBII_SDK_LOG_FILE	The log file to where the Tobii SDK logs information.

Windows Error Messages

Trying to run the sdk without having Bonjour installed can cause the following error. The easy fix is to install Bonjour for Windows. The installer is included in the SDK zip file.



If you run an SDK application but don't find any eye trackers, this may indicate that you have the wrong version of Bonjour installed. 64 bit applications written against the SDK requires the 64 bit version of Bonjour, and 32 bit SDK applications requires the 32 bit version of Bonjour. Both installers are included in the SDK zip file.

Appendices

Error codes

The following tables lists all currently defined error codes in the firmware and in the current SDK. Many of the herein listed error codes are internal and are listed for your reference.

Generic Errors

Name	Code (Decimal)	Code (Hex)	Description
TOBII_SDK_ERROR_SUCCESS	0	0	No error
TOBII_SDK_ERROR_GENERIC	1	1	Represents a generic error, i.e. no other suitable error exists
TOBII_SDK_ERROR_INVALID_ARGUMENTS	2	2	One or several of the arguments to a method/function were invalid. Typically occurs when one of the parameters is null.
TOBII_SDK_ERROR_OUT_OF_MEMORY	3	3	Not enough memory to perform a specific operation. Typically a severe error condition that is hard to handle.
TOBII_SDK_ERROR_OUT_OF_RANGE	4	4	One or several function/method arguments were out of range.
TOBII_SDK_ERROR_NOT_INITIALIZED	5	5	The SDK has not been initialized.
TOBII_SDK_ERROR_NOT_SUPPORTED	6	6	The called function is not supported.
TOBII_SDK_ERROR_TIMEOUT	30	0x0000001E	An outstanding async operation timed out. Can occur when connection to the eye tracker is lost.
TOBII_SDK_ERROR_OPERATION_ABORTED	32	0x0000001F	An outstanding async operation was aborted before it had time to complete.

Param Stack Errors

Name	Code (Decimal)	Code (Hex)	Description
TOBII_SDK_ERROR_INVALID_PAYLOAD_ITEM_TYPE	7	7	Occurs when the SDK application tries to serialize data of an unknown type.

Transport Errors

Name	Code (Decimal)	Code (Hex)	Description
TOBII_SDK_ERROR_TRANSPORT_ERROR	8	8	Internal Error
TOBII_SDK_ERROR_UNKNOWN_OPCODE	9	9	Internal Error
TOBII_SDK_ERROR_INVALID_PAYLOAD	10	0x0000000A	Internal Error. Message sent from the eye tracker contains invalid data
TOBII_SDK_ERROR_UNEXPECTED_PAYLOAD	11	0x0000000B	Internal Error. Message sent from the eye tracker

			contained data but none was expected
TOBII_SDK_ERROR_EMPTY_PAYLOAD	12	0x0000000C	Internal Error. Message sent from the eye tracker did not contain any data but data was expected.

Factory Info Errors

Name	Code (Decimal)	Code (Hex)	Description
TOBII_SDK_ERROR_INVALID_FACTORYINFO	20	0x00000014	Internal Error

Firmware Upgrade Errors

Name	Code (Decimal)	Code (Hex)	Description
TOBII_SDK_ERROR_UPGRADE_GENERIC	40	0x00000028	A generic error occurred during upgrade.
TOBII_SDK_ERROR_UPGRADE_SESSION_MISMATCH	41	0x00000029	Internal Error
TOBII_SDK_ERROR_UPGRADE_MISSING_PART_ID	42	0x0000002A	The upgrade package contains data that the SDK cannot understand.
TOBII_SDK_ERROR_UPGRADE_PACKAGE_VALIDATION	43	0x0000002B	The selected upgrade package file is not compatible with the selected eye tracker.
TOBII_SDK_ERROR_UPGRADE_WRONG_MODEL	44	0x0000002C	The selected upgrade package file is not compatible with the selected eye tracker. The package was made for another model.
TOBII_SDK_ERROR_UPGRADE_WRONG_GENERATION	45	0x0000002D	The selected upgrade package file is not compatible with the selected eye tracker. The package was made for another product generation.
TOBII_SDK_ERROR_UPGRADE_CANT_DOWNGRADE	46	0x0000002E	The installed firmware is newer than the firmware contained in the upgrade package file and downgrading is not supported.
TOBII_SDK_ERROR_UPGRADE_DEVICE_DATA_MISSING	47	0x0000002F	Failed to read data from the eye tracker when running compatibility checks.
TOBII_SDK_ERROR_OPERATION_ABORTED	32	0x0000001F	The firmware upgrade operation was

aborted.

Eye Tracker Errors

These are errors that are returned by the eye tracker in response to a request from the client.

Name	Code (Decimal)	Code (Hex)	Description
TOBII_FW_ERROR_SUCCESS	0	0	No error
TOBII_FW_ERROR_UNKNOWN_OPERATION	536872192	0x20000500	The specific opcode or request is unknown. This means that the eye tracker doesn't understand what it is supposed to do with this request.
TOBII_FW_ERROR_UNSUPPORTED_OPERATION	536872193	0x20000501	The eye tracker understands the opcode or request, but does not support it.
TOBII_FW_ERROR_OPERATION_FAILED	536872194	0x20000502	The request failed. This can mean different things for different requests.
TOBII_FW_ERROR_INVALID_PAYLOAD	536872195	0x20000503	The request contained invalid data. This is a severe error and indicates some kind of network problem.
TOBII_FW_ERROR_UNKNOWN_ID	536872196	0x20000504	The opcode referenced an unknown ID. This is probably caused by a programming error.
TOBII_FW_ERROR_UNAUTHORIZED	536872197	0x20000505	The operation cannot be completed without further authorization.
TOBII_FW_ERROR_EXTENSION_REQUIRED	536872198	0x20000506	The operation requires enabling an extension.
TOBII_FW_ERROR_INTERNAL_ERROR	536872190	0x20000507	A generic error that indicates a bug in the firmware.
TOBII_FW_ERROR_STATE_ERROR	536872200	0x20000508	The server or client is in a state where this request is unsupported or not allowed. This can for example occur when trying to call AddCalibrationPoint without first calling StartCalibration()
TOBII_FW_ERROR_INVALID_PARAMETER	536872201	0x20000509	One or more parameters in the request were incorrect.
TOBII_FW_ERROR_OPERATION_ABORTED	536872202	0x20000510	The request was aborted before it could complete. This indicates that the network connection to the eye tracker was lost.

