Tobii Gaze SDK

Developer's Guide

C API Specifics

This document is part of the developer's guide for the Tobii Gaze Software Development Kit (SDK). It provides information needed to develop gaze enabled applications using the C API.

Table of Contents

Introduction

This document gives a detailed description of the Tobii Gaze SDK C API.

It is recommended that you read the document *Tobii Gaze SDK Developer's Guide – General Concepts* before this document.

Supported development environments

The Tobii Gaze SDK works out-of-the-box with Visual Studio 2012 on Windows and with gcc on non-Windows platforms. The Gaze SDK can also be used with other versions of Visual Studio as long as the Visual C++ runtime libraries for VS2012 are present on the system. See the section on deployment below.

Contents of the Gaze SDK package

The Gaze SDK is distributed as archive files, one for each platform and runtime environment. If your application targets both Windows 32 and 64 bit platforms, you will need the redistributable libraries from both platforms. Everything else is identical in both platform packages.

Unzip the zip file to a directory of choice, e.g., "C:\tobiigazesdk". In the directory you will find the following subdirectories:

- docs – contains documentation for developers.
- include – contains the header files for the C API.
- lib – contains the libraries.
- samples – contains samples that demonstrate how to use the API. The samples are described in the section below.

C API header files

Including the tobiigaze.h header file will give you access to the most commonly used parts of the API. There are also a few special header files for advanced functionality in TobiiGazeCore: tobiigaze_calibration.h and tobiigaze_display_area.h.

There is no API reference in this document. It is available in the form of source code comments in the header files themselves.

Code samples

The included code samples demonstrate the use of the Tobii Gaze API. Note that the project files for the samples are compatible with Visual Studio 2012 and later versions. To build the samples using an earlier version of Visual Studio, you will need to create a new project file and add the source files manually.

MinimalTracker: This sample demonstrates the basic use of the TobiiGazeCore API to connect to an eye tracker, start the eye tracking, and receive gaze data packets. Connecting the eye tracker to the computer and running this sample with the `--auto` command line option should print the 2D gaze positions, for each eye individually, for 20 seconds before exiting. This sample uses the **synchronous**, i.e., blocking, API functions.

MinimalTrackerAsync: This sample is functionally identical to the MinimalTracker sample, but uses the **asynchronous**, i.e., non-blocking, API functions.

MinimalCalibration: This sample demonstrates the sequence of steps needed to calibrate the eye tracker. It cannot be used for calibration as is, because it is written as a console application and therefore cannot present any visual stimuli on the display, but it should be straightforward to implement the GUI part using your GUI framework of choice.

wxWidgetsCalibrationSample: This sample performs a user calibration of the eye tracker, including a basic positioning guide. It is written using an open source, cross-platform GUI library called wxWidgets. wxWidgets must be downloaded and configured separately: http://www.wxwidgets.org/

MFC sample (Windows only): This sample shows a trace of gaze points on the screen. *Note: MFC is the Microsoft Foundation Class library, which wraps portions of the Win32 API in C++ classes. More information can be found here: http://msdn.microsoft.com/en-us/library/d06h2x6e*

Building the code samples on Linux

Assuming that you have downloaded and unpacked the SDK package like so:

```
gunzip TobiiGazeSdk-CApi-<version>-linux64.tar.gz

tar -xvf TobiiGazeSdk-CApi-<version>-linux64.tar
```

The code samples can be built using the g++ compiler using the following commands:

```
cd TobiiGazeSdk-CApi-<version>-linux64/Samples

make
```

The makefile uses the rpath linker flag to embed the relative path to the tobiigazecore shared library in the executable file. This way of working is convenient during early development, but later in the process you'll probably want to install the library in a standard place and reference it from there. For more information about using shared libraries on Linux, see http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html .

Finally, take the sample program for a test run:

```
./tracker --auto
```

This should print the 2D gaze positions, for each eye individually, for 20 seconds.

Developing a gaze enabled application

A gaze enabled application can be written in any programming language that supports interfacing to C APIs, that is, virtually any modern programming language.

For a C or C++ application developed using Microsoft's Visual Studio development environment, the steps are as follows:

- Create a new Visual Studio project.
- Add the Gaze SDK include directory as an additional include directory.
- Add the Gaze SDK lib directory as an additional library directory.
- Add the TobiiGazeCore32/64.lib as additional dependencies. Specify the 32-bit or 64-bit variants according to the bitness of the build configuration.
- Copy the TobiiGazeCore32/64 dll files to the project output directory.

Make sure that you use the multi-threaded DLL variants of the C/C++ runtime libraries to avoid conflicts with the Gaze SDK libraries. This setting is available on the C/C++ code generation tab in the project properties dialog. Multi-threaded DLL is the default setting.

Make sure that you declare the calling convention properly for all functions that you pass as callbacks to the APIs. The APIs use the "__cdecl" calling convention throughout as defined by the TOBIIGAZE_CALL macro.

Strings are always passed as regular zero-terminated `char` strings in the API. The character encoding is UTF-8.

Deploying a gaze enabled application

Gaze enabled applications are typically installed using the Windows Installer, just like any desktop application.

The TobiiGazeCore32/64.dll should be installed in the application directory. They MUST NOT be installed in a system directory as it could cause compatibility issues with other installed gaze enabled applications. Also, please note that you need a redistribution agreement with Tobii to redistribute those libraries.

Windows only: Since the Gaze SDK dlls depend on the Visual C++ 2012 runtime libraries, it is recommended that the installers for those libraries are added as merge modules to the installer. If these libraries are not present on the computer, then applications that depend on them will refuse to start. The merge modules for the VC runtime libraries are installed with Visual Studio.

Sample session setup and tear down

This section describes the necessary steps of setting up and tearing down an eye tracking session. These steps are also shown in the MinimalTracker and MinimalTrackerAsync samples.

Setup

1. The first step is to call `tobiigaze_create` with an eye tracker URL to get a handle to an eye tracker. The eye tracker handle is used when calling other functions.
2. The next step is to start a thread and call `tobiigaze_run_event_loop` from that thread. You can use the `tobiigaze_run_event_loop_on_internal_thread` function instead if you do not want to create the thread yourself.
3. Connect to the eye tracker by calling `tobiigaze_connect`.
   Note: The connect call detects when the event loop thread has started. This means that it is perfectly safe to call connect directly after creating the event loop thread, without any additional thread synchronization.
4. At this point the basic setup setup is complete and there is an active connection with the eye tracker. It is now possible to start interacting with the eye tracker. At this point your application will typically call `tobiigaze_start_tracking` to start receiving gaze tracking data.

Tear down

1. To tear down the active connection to the eye tracker, call the function `tobiigaze_disconnect`.
2. The next step is to stop the event loop by calling `tobiigaze_break_event_loop`. This will make the blocking call to `tobiigaze_run_event_loop` return.
3. At this point it is important to wait for the event loop thread to terminate. This is usually done by calling `join`, or `WaitForSingleObject` on Windows, on the thread handle. If you used the `tobiigaze_run_event_loop_on_internal_thread` this will be taken care of automatically.
4. The final step is to call `tobiigaze_destroy` on the eye tracker handle. This step releases memory and performs other cleanup tasks.