Tobii Gaze SDK

Developer's Guide

.NET 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 .NET API.

Table of Contents

Introduction

This document describes the Tobii Gaze Software Development Kit (SDK) discussing .NET API specific classes, methods and structures.

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

Contents of the Gaze SDK package

The Gaze SDK is distributed as zip 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.
- lib – contains
    - dll files (both native and .NET)
    - XML help files (accessible from e.g. Visual Studio)
    - PDB files (for extended debugging information)
- Samples – contains samples that demonstrate how to use the API. The samples are described in the Code samples section below.

Code samples

The included code samples demonstrate the use of the Tobii Gaze APIs. 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 need to create a new project file from scratch.

MinimalTrackerNet: 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 methods. Note that there exist also asynchronous versions of all synchronous methods.

WinForms sample: This sample shows a trace of gaze points on the screen. This sample has a graphical user interface and is therefore probably the most relevant sample to start building your own application.

WPF calibration sample: This sample demonstrates how to calibrate the eye tracker. It is built as a Windows Presentation Foundation (WPF) application using the Model-View-ViewModel pattern.

Note: WinForms (or Windows Forms) is a framework for GUI development which is part of the .NET framework.

Developing a gaze enabled application

These are the steps needed to start working with the Gaze SDK in the Microsoft Visual Studio 2012 development environment:

- Create a new Visual Studio project.
- Open the Build/Configuration Manager dialog and change the active solution platform to x86 (the default is Any CPU). You will probably need to create a new solution platform to do this. The reason for explicitly specifying a bitness instead of using Any CPU is that you will need to use the matching TobiiGazeCore library, and if you use Any CPU, then the bitness of the process will depend on the bitness of the OS.
- Add the Tobii.Gaze.Core.Net.dll as a dependency.
- Add the TobiiGazeCore32/64.dll as additional dependencies. Specify the 32-bit or 64-bit variants according to the bitness of the build configuration.

Deploying a gaze enabled application

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

The TobiiGazeCore32/64.dll and Tobii.Gaze.Core.Net.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 be allowed to redistribute those libraries.

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 MinimalTrackerNet sample.

Setup

1. The first step is to create an EyeTracker instance (called `tracker` below) by specifying an eye tracker URL: `tracker = new EyeTracker(url)`.
2. The next step is to start a new thread and call `tracker.RunEventLoop()` from that thread. You can use the `tracker.RunEventLoopOnInternalThread(<completion status callback>)` method instead if you do not want to create the thread yourself.
3. Connect to the eye tracker by calling `tracker.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 start subscribing to gaze data by stating `tracker.GazeData += <your gaze data handler>` (can be done directly after creating the EyeTracker instance) and then call `tracker.StartTracking()` to start receiving gaze tracking data.

Tear down

1. To tear down the active connection to the eye tracker, call the method `tracker.Disconnect()`.
2. The next step is to stop the event loop by calling `tracker.BreakEventLoop()`. This will make the blocking call to `tracker.RunEventLoop()`return.
3. At this point it is important to wait for the event loop thread to terminate. This is done by calling `join()` on the previously created thread. If you used the `tracker.RunEventLoopOnInternalThread(…)` method this will be taken care of automatically.
4. The final step is to call dispose the EyeTracker instance by calling `tracker.Dispose()`. This step releases memory and performs other cleanup tasks.