



EyeLogic SDK

v 1.1.15

March 2025

1 EyeLogic SDK Documentation (C#)	1
1.1 Introduction	1
1.1.1 About	1
1.1.2 System Requirements	1
1.2 Installation and Getting Started	1
1.2.1 Download Software	1
1.2.1.1 Compatibility	2
1.2.2 Installing the EyeLogic SDK on Windows	2
1.2.3 Getting Started with the Sample Code	2
1.3 Concepts	3
1.3.1 Server-Client Setup	3
1.3.2 Set Up a Project for your Application	3
1.3.3 Control Flow between API and server	4
1.3.4 Dual PC Setup	4
1.3.5 Example Program	5
1.3.6 Gaze Samples	6
1.3.7 Shipping Your Application	6
1.4 Appendix	6
1.4.1 License Agreement and Warranty for SDK	6
1.5 About EyeLogic	8
1.5.1 Contact and Support	8
2 Namespace Index	9
2.1 Namespace List	9
3 Hierarchical Index	11
3.1 Class Hierarchy	11
4 Class Index	13
4.1 Class List	13
5 File Index	15
5.1 File List	15
6 Namespace Documentation	17
6.1 eyelogic Namespace Reference	17
6.1.1 Detailed Description	17
6.1.2 Enumeration Type Documentation	17
6.1.2.1 DeviceEventType	17
7 Class Documentation	19
7.1 ELCsApi.DeviceConfig Class Reference	19
7.1.1 Detailed Description	19
7.1.2 Member Data Documentation	19

7.1.2.1 deviceSerial	20
7.2 ELCsApi.DeviceGeometry Class Reference	20
7.2.1 Detailed Description	20
7.3 ELCsApi Class Reference	20
7.3.1 Detailed Description	22
7.3.2 Constructor & Destructor Documentation	22
7.3.2.1 ELCsApi()	22
7.3.3 Member Function Documentation	22
7.3.3.1 calibrate()	22
7.3.3.2 connect()	23
7.3.3.3 connectRemote()	23
7.3.3.4 getActiveScreen()	23
7.3.3.5 getAvailableScreens()	23
7.3.3.6 getDeviceConfig()	24
7.3.3.7 requestServerList()	24
7.3.3.8 requestTracking()	24
7.3.3.9 setActiveScreen()	25
7.3.3.10 unrequestTracking()	25
7.3.3.11 validate()	25
7.4 ELEException Class Reference	26
7.4.1 Detailed Description	26
7.4.2 Member Enumeration Documentation	26
7.4.2.1 ErrorType	26
7.5 GazeSample Class Reference	27
7.5.1 Detailed Description	27
7.5.2 Member Data Documentation	28
7.5.2.1 eyePositionLeft	28
7.5.2.2 eyePositionRight	28
7.6 Point2d Class Reference	28
7.6.1 Detailed Description	29
7.7 Point3d Class Reference	29
7.7.1 Detailed Description	29
7.8 ELCsApi.ScreenConfig Class Reference	29
7.8.1 Detailed Description	30
7.9 ELCsApi.ServerInfo Class Reference	30
7.9.1 Detailed Description	30
7.10 ELCsApi.ValidationPointResult Class Reference	30
7.10.1 Detailed Description	31
7.11 ELCsApi.ValidationResult Class Reference	31
7.11.1 Detailed Description	31

8.1 ELCsApi.cs File Reference	33
8.1.1 Detailed Description	34
Index	35

Chapter 1

EyeLogic SDK Documentation (C#)

1.1 Introduction

1.1.1 About

The EyeLogic Software Development Kit (SDK) is a free software package for building custom applications that use an EyeLogic eye tracking device. It provides the ability to connect to your device from any custom application via an Application Programming Interface (API). The EyeLogic SDK is available in the following programming languages C++, C#, C, and Python. It can also be used with any other programming language that can import dynamic link libraries (DLLs), such as Visual Basic or Matlab.

For each directly supported language, there is a short and simple example program to help you start developing your first EyeLogic application.

This manual describes how to use the EyeLogic API for C# and gives a step-by-step introduction on how to start with your own C# program.

1.1.2 System Requirements

Please refer to the EyeLogic Server documentation for system requirements and installation instructions.

The SDK has no additional requirements. The included C# example project is built for Microsoft Windows (32 bit or 64 bit) only. The sample projects are written for Microsoft Visual Studio 2019 or newer. Other compilers are not supported at this time.

1.2 Installation and Getting Started

1.2.1 Download Software

To use an EyeLogic eye tracking device from within your application, you need the EyeLogic Server and the EyeLogic SDK. Check the download page for the latest version of both packages: <https://www.eyelogicsolutions.com/downloads/>

1.2.1.1 Compatibility

The software is written to support backwards compatibility, i.e. updating the EyeLogic Server software will not break support for your device, regardless of the model. This guide assumes that you are installing the latest version of the EyeLogic Server. Please always update to the latest server version before reporting an error to the EyeLogic Support.

On the other hand, it is not always necessary to update the SDK and API DLLs. Since you as a programmer would have to recompile your application with each SDK update, we have designed the SDK to allow the server to communicate with older API versions. So when you ship your application, simply add the EyeLogic API DLLs of the current version to your package. It will be compatible with both current and newer versions of the server.

See [Shipping Your Application](#) for a tutorial on how to ship your application.

1.2.2 Installing the EyeLogic SDK on Windows

The EyeLogic SDK does not need to be installed. It is shipped as a .zip file that simply needs to be extracted to any directory on your hard drive. Make sure you have user rights to that directory, e.g. any directory within C:\Program Files or similar is problematic as it requires admin rights to access those files every time you start your program. It is recommended to use a local user directory.

Note: The SDK has to be installed on the same computer as the server. Please see the main server manual for help on installing the server.

After extracting the .zip file, the directory contains one subfolder for each supported programming language. Open the cs folder, the content should be:

- bin - contains the binary DLLs to link against
- example - contains the sample code

1.2.3 Getting Started with the Sample Code

In the directory, into which you unpacked the SDK EyeLogicSDK, navigate to the sub-directory cs/example and open the solution file AllDemoClients.sln in Visual Studio. Note, you will need Visual Studio 2017 or newer to open this file.

You may want to choose your destination compile level (Debug/Release) in the drop down list on top of the screen. Set it to "Debug" while developing your app. When your app is finished, set it to "Release" to create an optimized application binary. Then compile from the menu with Build->Build Solution. You should see output similar to the following:

```
Erstellen gestartet...
1>----- Erstellen gestartet: Projekt: DemoClient, Konfiguration: Debug x64 -----
2>----- Erstellen gestartet: Projekt: DualPC, Konfiguration: Debug x64 -----
3>----- Erstellen gestartet: Projekt: Validation, Konfiguration: Debug x64 -----
3> Validation -> cs\example\Validation\bin\x64\Debug\DemoClientCs.exe
2> DualPC -> cs\example\DualPC\bin\x64\Debug\DemoClientCs.exe
1> DemoClient -> cs\example\DemoClient\bin\x64\Debug\DemoClientCs.exe
1> ...\\EyeLogic_SDK\cs\example\..\bin\concr140.dll
1> ...\\EyeLogic_SDK\cs\example\..\bin\ELApi.dll
1> ...\\EyeLogic_SDK\cs\example\..\bin\ELCApi.dll
1> ...\\EyeLogic_SDK\cs\example\..\bin\ELCsApi.dll
1> ...\\EyeLogic_SDK\cs\example\..\bin\msvcpl140.dll
1> ...\\EyeLogic_SDK\cs\example\..\bin\msvcpl140_1.dll
1> ...\\EyeLogic_SDK\cs\example\..\bin\msvcpl140_2.dll
1> ...\\EyeLogic_SDK\cs\example\..\bin\vccorlib140.dll
1> ...\\EyeLogic_SDK\cs\example\..\bin\vcruntime140.dll
1> 9 Datei(en) kopiert.
===== Erstellen: 1 erfolgreich, 0 fehlerhaft, 0 aktuell, 0 übersprungen =====
```


Before running the application check that the EyeLogic Server is running (see the EyeLogic Server manual). If the server is running, there will be an EyeLogic icon in the Windows system tray.

On the left hand side of the Editor you will see a list of all projects/clients. The active one is highlighted in bold (DemoClient). You can make any other demo client active (e.g. DualPC or Validation) by right-clicking on the desired name in the list and setting it as the startup project.

Press F5 to compile and run the application.

Note that your firewall may block the connection between your program and the server. In this case, add a rule to your firewall to allow your application to open TCP/UDP ports to an application on localhost (for Windows Defender, just click "Accept").

If you have reached this point, you have set up your EyeLogic SDK correctly. You are now ready to start developing your own application. See the next section **Concepts** for the basic programming concepts and for a tutorial on how to deploy and ship your application.

1.3 Concepts

1.3.1 Server-Client Setup

The EyeLogic software consists of two main parts: The `server` and the `API`. The server is the necessary driver for your eye tracking device. It detects your device and handles the communication. The API is part of the EyeLogic Standard Development Kit (SDK). It consists of .dll files that can be used by your application to connect to the EyeLogic Server, start tracking and receive eye tracking data.

The server is designed to run continuously as a background process on your computer. When not actively tracking, the server uses a negligible amount of your computer's resources. Once an EyeLogic eye tracking device is connected, the Server application automatically detects it automatically and allows the user to set it up via the Server configuration dialogue (see the Server icon in the Windows tray bar). If for some reason the server background process is not running (the tray icon is missing), you can start the server manually from the Windows Start menu.

The API is a set of .dll files that can be used by any custom program (called a `user application`). These DLLs allow the user application to connect to the (running) server. Note that the EyeLogic Server can run on the same machine as the user application, or they can run on different PCs. See **Dual PC Setup** for how to set up the setup with running the server and the user application running on different machines.

1.3.2 Set Up a Project for your Application

For an easy start to developing a new application, it is recommended that you copy the existing sample folder to a new location (e.g. EyeLogic_SDK\cpp with all its contents). The sample source file already provides a fully functional implementation. From this sample code, you can easily modify and extend the code to suit your individual experiment.

Alternatively, you can start a new Visual Studio project from scratch. In this case, make sure that the compiler and linker are able to find the EyeLogic include and binary files. To do this, make the following changes to the project properties of your Visual Studio project:

- Select x64 as your target platform (it might require to add a new platform in your configuration manager)
 - In your project explorer, right click "references", click "add". In the following window select "browse" and browse to the file `<Location of your EyeLogic_SDK>\cs\bin\ELCsApi.dll`.
 - After compiling, copy all dll files from `<Location of your EyeLogic_SDK>\cs\bin` to your execution directory.
-

1.3.3 Control Flow between API and server

The usual control flow between the custom application/API and the server is characterised by the following steps:

1. **initialize:** Before calling any other function the API DLLs need initializing.
2. **connect to server:** Establish a connection to the server via TCP.
3. **find eye tracking device:** Obtain the information on connected eye trackers, otherwise wait until an eye tracker is plugged in.
4. **start tracking:** Request tracking. If successful, the device will start tracking and the server sends *Gaze Samples* to the user application, see also [Gaze Samples](#).
5. **perform calibration:** Request a calibration. A calibration point will appear on the screen, animated to move across the screen. The user must fixate on this point until the calibration screen disappears. The system is calibrated and ready for use when this process is successfully completed.
6. **shut down:** At the end of your experiment either stop the tracking or simply shutdown the API.

All information which is passed from the server to the user application is passed via *asynchronous callbacks*. The application must register its own implementations of these callback functions with the API (see [Example Program](#) for a sample implementation).

Note that you need to calibrate to get valid gaze samples (see [Gaze Samples](#)). Any gaze samples reported before the system is calibrated will not contain valid eye data.

1.3.4 Dual PC Setup

The Dual PC Setup is a special setup where the EyeLogic Server runs on a different computer than the user application.

The most common use case for the Dual PC Setup would be the following. Running an experiment with an operator controlling the eye tracking device and a participant performing a task. The participant uses a different PC (which displays the experiment) than the operator (who can control the eye tracker via the EyeLogic Server software).

The operator's computer (called the Operator PC) must have the EyeLogic driver software (the EyeLogic Server) installed and running. The eye tracker is physically attached to a monitor that is connected to the participant's computer (called the Experiment PC). The USB cable of the eye tracker is plugged into the USB port of the Operator PC!

The operator can now use the server to detect the eye tracking device. On the Experiment PC, any custom application that presents an experiment to the participant can use the EyeLogic API to remotely connect to the server. To do this, the application should use the API calls:

1. `elRequestServerList()` to obtain a list of all EyeLogic servers in the local network (LAN/WLAN) which are running and are configured to allow remote connections
2. `elConnectRemote()` to connect to a specific server from that list
3. `elSetActiveScreen()` to set the screen connected to the Experiment PC as the active screen for eye tracking (replacing the default main screen of the Operator PC)

Note, that a server must allow remote connections for it to be found. You can enable this in the settings of the server window.

If the connection is successful, the client can operate as usual as if it were connected to a local server. See the demo application "dualpc" demo application in the SDK for an example.

1.3.5 Example Program

In this section, the code of the C# example program is explained in some detail.

The file starts with an include section. The important include is

```
using eyelogic;
```

which is needed to find all necessary definitions of the EyeLogic API.

In the `run()` method the application implements its control flow. It consists of the following code lines:

```
m_api = new ELCsApi( "C# Client" );
```

This constructs a new instance of the `ELCsApi` class. The instantiation will automatically initialize the library. The API needs to be initialized only once throughout the whole program. At the end, when finished, deinitialize the API with

```
m_api.destroy( );
```

The next two lines

```
m_api.OnEvent += onEvent;
m_api.OnGazeSample += onGazeSample;
```

register the callback functions which are invoked from the EyeLogic software whenever an event occurs and whenever a new gaze sample is incoming. Those functions are defined further below. The example code simply prints an incoming event to the main console, resp. count the incoming gaze samples.

```
m_api.connect( );
```

Connects to the EyeLogic server. If the connection fails, an `ELException` is thrown. If the method exits without an exception, then the connection is established.

```
ELCsApi.DeviceConfig deviceConfig = m_api.getDeviceConfig( );
```

obtains information about the connected eye tracking device. If there is no device connected, the method returns null.

```
ELCsApi.ScreenConfig screenConfig = m_api.getActiveScreen( );
```

obtains information about the active screen.

```
m_api.requestTracking( 0 );
```

Tells the device to start tracking and the Server to start sample processing. Parameter 0 specifies the frame rate mode. If your device is capable of multiple frame rate modes (60Hz, 120Hz or 250Hz), you can specify a different number. The list of available frame rate modes is part of the `DeviceConfig` and can be obtained by calling `getDeviceConfig()`. The first frame rate mode (`DeviceConfig.frameRates[0]`) is the default mode, which is usually the highest available speed mode on your system.

```
const auto retCalibrate = api.calibrate( 0 );
```

Performs a calibration. This method blocks until the calibration is finished - i.e. completed or cancelled. The parameter 0 indicates the type of calibration. A list of available calibration methods is part of the `DeviceConfig` and can be obtained by calling `getDeviceConfig()`.

The example program waits for 5 seconds and then closes the connection:

```
m_api.unrequestTracking( );
m_api.disconnect( );
m_api.destroy( );
```

The last two lines unregister the callback functions. Be sure to unregister them before destroying the API object.

1.3.6 Gaze Samples

Gaze samples are the most important data which is generated by the eye tracker. The eye tracker provides one gaze sample per frame. Each sample contains information about the time of measurement, the position of the eyes, the pupil radius and the point at which the user is looking on a stimulus plane (usually a computer monitor).

1.3.7 Shipping Your Application

When you want to ship your application, be sure to include all relevant files so that it can run on different computers. The EyeLogic functionality will only work on computers that have the EyeLogic Server installed. The installed server must be at least be of the same version as the supplied API DLLs (a newer server version is acceptable).

In addition to the relevant files of your application, you need to ship the contents of the bin/ folder of your language (typically including some .dll files). Place the contents of the bin/ folder in the working directory of your application and ship them together.

1.4 Appendix

1.4.1 License Agreement and Warranty for SDK

IMPORTANT – PLEASE READ CAREFULLY:

The License Agreement is a legal agreement between you and EyeLogic GmbH and its affiliates (“EyeLogic”, “we”, or “us”). This license agreement governs your use of the EyeLogic software and any third party software that may be distributed therewith (collectively the “software”). EyeLogic agrees to license the software to you (personally and/or on behalf of you employer) (collectively “you” or “your”) only if you accept all the terms contained in this license agreement. By installing, using, copying, or distributing all or any portion of the software, you accept and agree to be bound by all of the terms and conditions of this license agreement.

If you do not agree with any of the terms of this license agreement, do not install or use the software.

1. **License Grant:** EyeLogic grants you a revocable, nonexclusive, non-transferable, limited right to install and use the application on a device owned and controlled by you, and to access and use the application on such mobile device strictly in accordance with the terms and conditions of this licenses, the usage rules and any service agreement associated with your device. The Software includes third party software and other copyrighted material. Acknowledgements, licensing terms and disclaimers for such Third Party Software are provided with the Software or contained in the Documentation, and your use of such Third Party Software is governed by their respective terms (collectively “Related Agreements”).
 2. **Restriction on Use:** You shall use the application strictly in accordance with the terms of the related agreements and shall not:
 - (a) decompile, reverse engineer, disassemble, attempt to derive the source code of, or decrypt the application,
 - (b) make any modification, adaption, improvement, enhancement, translation or derivative work from the application,
 - (c) violate any applicable laws, rules or regulations in connection with your access or use of the application,
 - (d) remove, alter or obscure any proprietary notice (including any notice of copyright or trademark) of EyeLogic or its affiliates, partners, suppliers or the licensors of the application,
 - (e) use the application for any revenue generating endeavor, commercial enterprise or other purpose for which it is not designed or intended,
-

- (f) make the application publicly available over a network or other environment permitting access or use by others without the written permission of EyeLogic,
 - (g) use the application for creating a product, service or software that is, directly or indirectly, competitive with or in any way substitute for any services, product or software offered by EyeLogic,
 - (h) use any proprietary information or interfaces of EyeLogic or other intellectual property of EyeLogic in the design, development, manufacture, licensing or distribution of any applications, accessories or devices for use with the application.
3. **Termination:** EyeLogic may, in its sole and absolute discretion, at any time and for any or no reason, suspend or terminate this license and the rights afforded to you hereunder with or without prior notice. Furthermore, if you fail to comply with any terms and conditions of this license, then this license and any rights afforded to you hereunder shall terminate automatically, without any notice or other action by EyeLogic. Upon the termination of this license, you shall cease all use of the application and uninstall the application.
4. **Disclaimer of Warranties:** You acknowledge and agree that the application is provided on an “as is” and “as available” basis, and that your use of or reliance upon the application and any third party content and services accessed thereby is at your sole risk and discretion. EyeLogic and its affiliates, partners, suppliers and licensors hereby disclaim any and all representations, warranties and guarantees regarding the application and third party content and services, whether express, implied or statutory, and including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, furthermore, EyeLogic and its affiliates, partners, suppliers and licensors make no warranty that
- (a) The application or third party content and services will meet your requirements,
 - (b) The application or third party content and services will be uninterrupted, accurate, reliable, timely, secure or error-free,
 - (c) The quality of any products, services, information or other material accessed or obtained by you through the application will be as represented or meet your expectations, or
 - (d) Any errors in the application or third party content and services will be corrected.
- No advice or information, whether oral or written, obtained by you from EyeLogic or from the application will create any warranty not expressly made herein or create any liability on the part of EyeLogic.
- If the licensee modifies or replaces any of the third party open source software included in the software, EyeLogic is not obligated to provide any updates, maintenance, warranty, technical or other support or services for the resultant modified Software. You expressly acknowledge that any failure or damage to any hardware, software or systems as a result of such modification to the open source components of the software is excluded from the terms of any EyeLogic warranty.
5. **Limitation of liability:** Under no circumstances shall EyeLogic or its affiliates, partners, suppliers or licensors be liable for any indirect, incidental, consequential, special or exemplary damages arising out of or in connection with your access or use of or inability to access or use the application and any third party content and services, whether or not the damages were foreseeable and whether or not EyeLogic was advised of the possibility of such damages. Without limiting the generality of the foregoing, EyeLogic's aggregate liability to you (whether under contract, tort, statute or otherwise) shall not exceed the amounts actually paid by licensee for the licensed materials. The foregoing limitations will apply even if the above stated remedy fails of its essential purpose.
6. **Confidentiality:** Licensed materials are proprietary to EyeLogic and constitute EyeLogic trade and business secrets. The licensee shall maintain licensed materials in confidence and prevent their disclosure using at least the same degree of care it uses for its own trade and business secrets, but in no event less than a reasonable degree of care. The licensee shall not disclose licensed materials or any part thereof to anyone for any purpose, other than to its employees and sub-contractors, if any, for the purpose of exercising the rights expressly granted under this agreement, provided they have in writing agreed to confidentiality obligations at least equivalent to the obligations stated herein. The foregoing does not apply to information that a. is or becomes generally known or available to the public without any breach of the confidentiality obligation by licensee, b. was already known to licensee prior to the disclosure by EyeLogic, or c. was rightfully acquired by licensee from a third party without a breach of a confidentiality obligation towards EyeLogic. In case of a dispute, the licensee has the burden of proof that the licensed materials and/or any portion thereof fall under one of these exceptions. Should the licensee be legally compelled to disclose any licensed materials to a third party, such as pursuant to a mandatory order by a court or authority or any comparable action, the licensee
-

shall, to the extent permitted under applicable law, inform EyeLogic without undue delay and undertake all possible measures to safeguard secrecy.

1.5 About EyeLogic

EyeLogic is a manufacturer of high precision and high quality eye tracking devices, mainly for scientific and research use cases. EyeLogic GmbH is a spin-off of the Free University of Berlin, faculty of mathematics and computer science and has a vast experience in image processing and computer vision.

1.5.1 Contact and Support

For technical support questions contact us via mail at: support@eyelogicsolutions.com

EyeLogic GmbH
Schlesische Str. 28
10997 Berlin Germany
www: <https://www.eyelogicsolutions.com>

Copyright © EyeLogic GmbH

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

eyelogic	
Namespace for C# API calls	17

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ELCsApi.DeviceConfig	19
ELCsApi.DeviceGeometry	20
ELCsApi	20
Exception	
ELException	26
GazeSample	27
Point2d	28
Point3d	29
ELCsApi.ScreenConfig	29
ELCsApi.ServerInfo	30
ELCsApi.ValidationPointResult	30
ELCsApi.ValidationResult	31

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ELCsApi.DeviceConfig	
Device configuration	19
ELCsApi.DeviceGeometry	
Geometric position of the device related to the active monitor	20
ELCsApi	
Main class for communication with the EyeLogic server	20
ELException	
EyeLogic Exception class. API functions may throw this exception, catch it for error handling	26
GazeSample	
EyeLogic GazeSample	27
Point2d	
2D point	28
Point3d	
3D point	29
ELCsApi.ScreenConfig	
Screen configuration	29
ELCsApi.ServerInfo	
Connection information for an EyeLogic server	30
ELCsApi.ValidationPointResult	
ValidationPointResult holds the results of the validation (total deviation between true point position and calculated POR of the left and right eye POR in [px] and [deg]) of the validation point at position (validationPointPxX, validationPointPxY) [px]	30
ELCsApi.ValidationResult	
ValidationResult contains one ValidationPointResult struct per validation stimulus point of the performed validation. ValidationPointResult data fields may be ELCsLib.InvalidValue	

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

ELCsApi.cs

The file contains the C# definitions which are necessary to control the EyeLogic software from an API client

33

Chapter 6

Namespace Documentation

6.1 eyelogic Namespace Reference

namespace for C# API calls

Classes

- class **ELCsApi**
main class for communication with the EyeLogic server
- class **ELException**
EyeLogic Exception class. API functions may throw this exception, catch it for error handling.
- class **GazeSample**
EyeLogic GazeSample.
- class **Point2d**
2D point
- class **Point3d**
3D point

Enumerations

- enum **DeviceEventType** {
 SCREEN_CHANGED, **CONNECTION_CLOSED**, **DEVICE_CONNECTED**, **DEVICE_DISCONNECTED**,
 TRACKING_STOPPED }
Device events.

6.1.1 Detailed Description

namespace for C# API calls

6.1.2 Enumeration Type Documentation

6.1.2.1 DeviceEventType

```
enum DeviceEventType [strong]
```

Device events.

Enumerator

SCREEN_CHANGED	a new screen has been set as active
CONNECTION_CLOSED	connection to EyeLogic Server has closed
DEVICE_CONNECTED	a new device has connected
DEVICE_DISCONNECTED	device disconnected
TRACKING_STOPPED	tracking has stopped

Chapter 7

Class Documentation

7.1 ELCsApi.DeviceConfig Class Reference

device configuration

Public Member Functions

- string **formatDeviceSerial** ()
get device serial number as formatted string

Public Attributes

- ulong **deviceSerial**
serial number of the device as unsigned 64-bit int for a verbose format, call
- string **deviceName**
name of the device
- string **brandedName**
name of the license owner
- bool **isDemoDevice**
whether the device is for DEMO use only, not for public sale
- List< int > **frameRates**
list of available framerates [Hz]
- List< int > **calibrationMethods**
list of available calibration methods [number of calibration points]

7.1.1 Detailed Description

device configuration

7.1.2 Member Data Documentation

7.1.2.1 deviceSerial

```
ulong deviceSerial
```

serial number of the device as unsigned 64-bit int for a verbose format, call

See also

[formatDeviceSerial](#)

7.2 ELCsApi.DeviceGeometry Class Reference

Geometric position of the device related to the active monitor.

Public Attributes

- double [mmBelowScreen](#)
vertical distance between the lowest pixel on the display and the upper edge of the eye tracker
- double [mmTrackerInFrontOfScreen](#)
horizontal distance between the front of the screen and the front edge of the eye tracker

7.2.1 Detailed Description

Geometric position of the device related to the active monitor.

7.3 ELCsApi Class Reference

main class for communication with the EyeLogic server

Classes

- class [DeviceConfig](#)
device configuration
- class [DeviceGeometry](#)
Geometric position of the device related to the active monitor.
- class [ScreenConfig](#)
screen configuration
- class [ServerInfo](#)
connection information for an EyeLogic server
- class [ValidationPointResult](#)
[ValidationPointResult](#) holds the results of the validation (total deviation between true point position and calculated POR of the left and right eye POR in [px] and [deg]) of the validation point at position (validationPointPxX, validationPointPxY) [px].
- class [ValidationResult](#)
[ValidationResult](#) contains one [ValidationPointResult](#) struct per validation stimulus point of the performed validation. [ValidationPointResult](#) data fields may be [ELCsLib.InvalidValue](#)

Public Member Functions

- delegate void **ELDeviceEvent** (DeviceEventType id)
Event type.
- delegate void **ELGazeSample** (GazeSample sample)
GazeSample event type.
- delegate void **ELEyeImage** (Bitmap eyeImage)
EyeImage event type.
- **ELCsApi** (string clientName)
constructor
- void **destroy** ()
destroys the ELCsApi. Call this once before shutting down.
- void **connect** ()
initialize connection to the server (method is blocking until connection established). The connection is only established for a local server (running on this machine). For connections to a remote server,
- void **connectRemote** (ServerInfo server)
initialize connection to a remote server (method is blocking until connection established)
- **ServerInfo[] requestServerList** (int blockingDurationMS, int maxNumServer)
Ping all running EyeLogic servers in the local network and wait some time for their response.
- void **disconnect** ()
closes connection to the server
- bool **isConnected** ()
whether a connection to the server is established
- **ScreenConfig getActiveScreen** ()
obtain configuration of active screen
- **ScreenConfig[] getAvailableScreens** ()
Get a list of screens connected to the local machine. If there are more screens than 'numScreenConfigs' found, then only the first 'numScreenConfigs' ones are filled.
- void **setActiveScreen** (string screenID, DeviceGeometry deviceGeometry)
Make a screen connected to this machine to the active screen.
- **DeviceConfig getDeviceConfig** ()
obtain configuration of active device
- void **streamEyeImages** (bool enable)
- void **requestTracking** (int frameRateModelInd)
request tracking
- void **unrequestTracking** ()
unrequest tracking
- void **calibrate** (int calibrationModelInd)
perform calibration (method is blocking until calibration finished)
- **ValidationResult validate** ()
perform validation (method is blocking until validation finished) - calibration must be performed prior

Static Public Attributes

- static double **InvalidValue** = Double.MinValue
marker for an invalid double value

Events

- `ELDeviceEvent OnDeviceEvent = delegate { }`
Event, add your event handler to become notified on new events.
- `ELGazeSample OnGazeSample = delegate { }`
GazeSample event, add your event handler to become notified on new gaze samples.
- `ELEyeImage OnEyeImage = delegate { }`
EyeImage event, add your event handler to become notified on new eye images.

7.3.1 Detailed Description

main class for communication with the EyeLogic server

7.3.2 Constructor & Destructor Documentation

7.3.2.1 ELCsApi()

```
ELCsApi (
    string clientName ) [inline]
```

constructor

Parameters

<i>clientName</i>	string identifier of the client (shown in the server tool window), may be null
-------------------	--

7.3.3 Member Function Documentation

7.3.3.1 calibrate()

```
void calibrate (
    int calibrationModeInd ) [inline]
```

perform calibration (method is blocking until calibration finished)

Parameters

<i>calibrationModeInd</i>	index of the requested calibration method in <code>DeviceConfig.calibrationMethods</code>
---------------------------	---

7.3.3.2 connect()

```
void connect ( ) [inline]
```

initialize connection to the server (method is blocking until connection established). The connection is only established for a local server (running on this machine). For connections to a remote server,

See also

[connectRemote\(\)](#).

7.3.3.3 connectRemote()

```
void connectRemote (
    ServerInfo server ) [inline]
```

initialize connection to a remote server (method is blocking until connection established)

Parameters

<i>server</i>	Server to connect to
---------------	----------------------

See also

[acquireServerList\(\)](#) to obtain IP address and port of a remote server

7.3.3.4 getActiveScreen()

```
ScreenConfig getActiveScreen ( ) [inline]
```

obtain configuration of active screen

Returns

[ScreenConfig](#) returns the config of the active screen

7.3.3.5 getAvailableScreens()

```
ScreenConfig [ ] getAvailableScreens ( ) [inline]
```

Get a list of screens connected to the local machine. If there are more screens than 'numScreenConfigs' found, then only the first 'numScreenConfigs' ones are filled.

Parameters

<i>screenConfig</i>	pre-allocated array, will be filled with screen configurations
<i>numScreenConfigs</i>	number of entries of screenConfig

Returns

number of filled screen configurations. will be \leq numScreenConfigs

7.3.3.6 getDeviceConfig()

```
DeviceConfig getDeviceConfig ( ) [inline]
```

obtain configuration of active device

Returns

DeviceConfig returns the config of the connected device

7.3.3.7 requestServerList()

```
ServerInfo [ ] requestServerList (
    int blockingDurationMS,
    int maxNumServer ) [inline]
```

Ping all running EyeLogic servers in the local network and wait some time for their response.

Parameters

<i>blockingDurationMS</i>	waiting duration in milliseconds. Method returns at the latest after this time.
<i>maxNumServer</i>	Maximum number of server to be searched. Method returns immediately when that number of server is found.

Returns

List of all responding EyeLogic servers

7.3.3.8 requestTracking()

```
void requestTracking (
    int frameRateModeInd ) [inline]
```

request tracking

If tracking is not yet ongoing, tracking is started in the device. If tracking is already running (e.g. started from another client) with the same frame-rate as requested, all gaze samples are reported to this client as well.

Parameters

<i>frameRateModelInd</i>	index of the requested frame rate mode in DeviceConfig.frameRates
--------------------------	---

7.3.3.9 setActiveScreen()

```
void setActiveScreen (
    string screenID,
    DeviceGeometry deviceGeometry ) [inline]
```

Make a screen connected to this machine to the active screen.

Recording is from now on performed on the new active screen. Remember to perform a calibration on the new screen, otherwise it remains in an uncalibrated state.

Parameters

<i>screenID</i>	ID of the new active screen on <i>this</i> machine (even works if the connection to the server is remote). If null, the primary screen of this machine is set as active.
<i>deviceGeometry</i>	Geometry of the device which is mounted to the screen.

Returns

success/error code

7.3.3.10 unrequestTracking()

```
void unrequestTracking ( ) [inline]
```

unrequest tracking

Note that the tracking device may continue if other processes still request tracking. Check the EyeLogic server window to observe the actual state.

7.3.3.11 validate()

```
ValidationResult validate ( ) [inline]
```

perform validation (method is blocking until validation finished) - calibration must be performed prior

Returns

[ValidationResult](#)

7.4 EException Class Reference

EyeLogic Exception class. API functions may throw this exception, catch it for error handling.

Public Types

- enum `ErrorType` {
UNKNOWN_ERROR, ALREADY_INITED, NOT_INITED, VERSION_MISMATCH,
CONNECTION_FAILED, NOT_CONNECTED, DEVICE_MISSING, INVALID_FRAMERATE_MODE,
ALREADY_TRACKING, NOT_TRACKING, INVALID_CALIBRATION_MODE, ALREADY_CALIBRATING_OR_VALIDATING,
NOT_CALIBRATED, SCREEN_NOT_FOUND, SCREEN_FAILURE }
Error type.

Public Member Functions

- `EException ()`
Default constructor.
- `EException (ErrorType error, string message)`
Constructor with error type and message string.

Public Attributes

- `ErrorType Error`
error type

7.4.1 Detailed Description

EyeLogic Exception class. API functions may throw this exception, catch it for error handling.

7.4.2 Member Enumeration Documentation

7.4.2.1 ErrorType

```
enum ErrorType [strong]
```

Error type.

Enumerator

UNKNOWN_ERROR	not specified
ALREADY_INITED	cannot initialize library: was already initialized before
NOT_INITED	library not correctly initialized
VERSION_MISMATCH	connection failed: API is build on a newer version than the server. Update the EyeLogicServer to the newest version.

Enumerator

CONNECTION_FAILED	connection failed: the server can not be found or is not responding
NOT_CONNECTED	not connected to the server
DEVICE_MISSING	cannot start tracking: no device found
INVALID_FRAMERATE_MODE	cannot start tracking: framerate mode is invalid or not supported
ALREADY_TRACKING	tracking already ongoing, but frame rate mode is different
NOT_TRACKING	cannot calibrate: no device found or tracking not started
INVALID_CALIBRATION_MODE	cannot start calibration: calibration mode is invalid or not supported
ALREADY_CALIBRATING_OR_VALIDATING	cannot start calibration or validation: a calibration or validation is already in progress
NOT_CALIBRATED	cannot start validation: device must be calibrated
SCREEN_NOT_FOUND	cannot set active screen: given screen was not found
SCREEN_FAILURE	cannot set active screen

7.5 GazeSample Class Reference

EyeLogic [GazeSample](#).

Public Attributes

- long [timestampMicroSec](#)
timepoint when data was acquired in microseconds after EPOCH
- int [index](#)
increasing [GazeSample](#) index
- [Point2d](#) [porRaw](#)
binocular point of regard on the stimulus plane, check [porRaw.x](#) != [ELCsAPI.InvalidValue](#) before using it
- [Point2d](#) [porFiltered](#)
filtered binocular point of regard on the stimulus plane, check [porFiltered.x](#) != [ELCsAPI.InvalidValue](#) before using it
- [Point2d](#) [porLeft](#)
monocular point of regard of the left eye, check [porLeft.x](#) != [ELCsAPI.InvalidValue](#) before using it
- [Point3d](#) [eyePositionLeft](#)
position of the left eye in device coordinates, unit is mm
- double [pupilRadiusLeft](#)
radius of the left pupil in mm or [ELCsAPI.InvalidValue](#) if eye was not found
- [Point2d](#) [porRight](#)
monocular point of regard of the right eye, check [porRight.x](#) != [ELCsAPI.InvalidValue](#) before using it
- [Point3d](#) [eyePositionRight](#)
position of the right eye in device coordinates, unit is mm
- double [pupilRadiusRight](#)
radius of the right pupil in mm or [ELCsAPI.InvalidValue](#) if eye was not found

7.5.1 Detailed Description

EyeLogic [GazeSample](#).

7.5.2 Member Data Documentation

7.5.2.1 eyePositionLeft

`Point3d eyePositionLeft`

position of the left eye in device coordinates, unit is mm

- (0, 0, 0) is in the center of the device
- x-coordinate: positive towards the right side of the screen
- y-coordinate: positive towards the top of the screen
- z-coordinate: distance in front of the screen

check `eyePositionLeft.x != ELCsAPI.InvalidValue` before using it

7.5.2.2 eyePositionRight

`Point3d eyePositionRight`

position of the right eye in device coordinates, unit is mm

- (0, 0, 0) is in the center of the device
- x-coordinate: positive towards the right side of the screen
- y-coordinate: positive towards the top of the screen
- z-coordinate: distance in front of the screen

check `eyePositionRight.x != ELCsAPI.InvalidValue` before using it

7.6 Point2d Class Reference

2D point

Public Member Functions

- `Point2d` (double `x`, double `y`)
Constructor.

Public Attributes

- double **x**
x coordinate of the point
- double **y**
y coordinate of the point

7.6.1 Detailed Description

2D point

7.7 Point3d Class Reference

3D point

Public Member Functions

- **Point3d** (double **x**, double **y**, double **z**)
Constructor.

Public Attributes

- double **x**
x coordinate of the point
- double **y**
y coordinate of the point
- double **z**
z coordinate of the point

7.7.1 Detailed Description

3D point

7.8 ELCsApi.ScreenConfig Class Reference

screen configuration

Public Attributes

- bool **localMachine**
whether the screen is connected to the this machine
- string **id**
identifier name of the screen
- string **name**
descriptive name of the screen
- int **resolutionX**
screen resolution [px]
- int **resolutionY**
screen resolution [px]
- double **physicalSizeX_mm**
physical dimension of the screen [mm]
- double **physicalSizeY_mm**
physical dimension of the screen [mm]

7.8.1 Detailed Description

screen configuration

7.9 ELCsApi.ServerInfo Class Reference

connection information for an EyeLogic server

Public Attributes

- string **ip**
IP address of server as 0-terminated string.
- ushort **port**
port of server

7.9.1 Detailed Description

connection information for an EyeLogic server

7.10 ELCsApi.ValidationPointResult Class Reference

ValidationPointResult holds the results of the validation (total deviation between true point position and calculated POR of the left and right eye POR in [px] and [deg]) of the validation point at position (**validationPointPxX**, **validationPointPxY**) [px].

Public Attributes

- double **validationPointPxX**
x-coordinate of stimulus point position
- double **validationPointPxY**
y-coordinate of stimulus point position
- double **meanDeviationLeftPx**
InvalidValue or mean deviation between left eye POR and stimulus position in [px] in the stimulus plane.
- double **meanDeviationLeftDeg**
InvalidValue or mean deviation of left eye gaze direction in [deg] in the 3-D world system.
- double **meanDeviationRightPx**
InvalidValue or mean deviation between right eye POR and stimulus position in [px] in the stimulus plane.
- double **meanDeviationRightDeg**
InvalidValue or mean deviation of right eye gaze direction in [deg] in the 3-D world system.

7.10.1 Detailed Description

ValidationPointResult holds the results of the validation (total deviation between true point position and calculated POR of the left and right eye POR in [px] and [deg]) of the validation point at position (**validationPointPxX**, **validationPointPxY**) [px].

The stimulus point position and deviation [px] are given in the 2D stimulus coordinate system originating in the top left corner of the stimulus.

The deviation [deg] corresponds to the total angular deviation between the measured gaze direction from the ground truth gaze direction as determined according to the measured eye position.

Note: meanDeviation* data fields may be **ELCsApi.InvalidValue**. meanDeviationLeftDeg/-Px and meanDeviation↔RightDeg/-Px are always either both valid or both **ELCsApi.InvalidValue**.

7.11 ELCsApi.ValidationResult Class Reference

ValidationResult contains one **ValidationPointResult** struct per validation stimulus point of the performed validation. **ValidationPointResult** data fields may be **ELCsLib.InvalidValue**

Public Attributes

- List< **ValidationPointResult** > **pointsData**

7.11.1 Detailed Description

ValidationResult contains one **ValidationPointResult** struct per validation stimulus point of the performed validation. **ValidationPointResult** data fields may be **ELCsLib.InvalidValue**

Chapter 8

File Documentation

8.1 ELCsApi.cs File Reference

The file contains the C# definitions which are necessary to control the EyeLogic software from an API client.

Classes

- class **Point2d**
2D point
- class **Point3d**
3D point
- class **GazeSample**
EyeLogic GazeSample.
- class **ELException**
EyeLogic Exception class. API functions may throw this exception, catch it for error handling.
- class **ELCsApi**
main class for communication with the EyeLogic server
- class **ELCsApi.ServerInfo**
connection information for an EyeLogic server
- class **ELCsApi.ScreenConfig**
screen configuration
- class **ELCsApi.DeviceGeometry**
Geometric position of the device related to the active monitor.
- class **ELCsApi.DeviceConfig**
device configuration
- class **ELCsApi.ValidationPointResult**
ValidationPointResult holds the results of the validation (total deviation between true point position and calculated POR of the left and right eye POR in [px] and [deg]) of the validation point at position (validationPointPxX, validation←→ PointPxY) [px].
- class **ELCsApi.ValidationResult**
ValidationResult contains one ValidationPointResult struct per validation stimulus point of the performed validation. ValidationPointResult data fields may be ELCsLib.InvalidValue

Namespaces

- namespace `eyelogic`
namespace for C# API calls

Enumerations

- enum `DeviceEventType` {
 `SCREEN_CHANGED`, `CONNECTION_CLOSED`, `DEVICE_CONNECTED`, `DEVICE_DISCONNECTED`,
 `TRACKING_STOPPED` }
Device events.

8.1.1 Detailed Description

The file contains the C# definitions which are necessary to control the EyeLogic software from an API client.

Index

ALREADY_CALIBRATING_OR_VALIDATING

 EException, 27

ALREADY_INITED

 EException, 26

ALREADY_TRACKING

 EException, 27

calibrate

 ELCsApi, 22

connect

 ELCsApi, 22

CONNECTION_CLOSED

 eyelogic, 18

CONNECTION_FAILED

 EException, 27

connectRemote

 ELCsApi, 23

DEVICE_CONNECTED

 eyelogic, 18

DEVICE_DISCONNECTED

 eyelogic, 18

DEVICE_MISSING

 EException, 27

DeviceEventType

 eyelogic, 17

deviceSerial

 ELCsApi.DeviceConfig, 19

ELCsApi, 20

 calibrate, 22

 connect, 22

 connectRemote, 23

 ELCsApi, 22

 getActiveScreen, 23

 getAvailableScreens, 23

 getDeviceConfig, 24

 requestServerList, 24

 requestTracking, 24

 setActiveScreen, 25

 unrequestTracking, 25

 validate, 25

ELCsApi.cs, 33

ELCsApi.DeviceConfig, 19

 deviceSerial, 19

ELCsApi.DeviceGeometry, 20

ELCsApi.ScreenConfig, 29

ELCsApi.ServerInfo, 30

ELCsApi.ValidationPointResult, 30

ELCsApi.ValidationResult, 31

EException, 26

 ALREADY_CALIBRATING_OR_VALIDATING, 27

 ALREADY_INITED, 26

 ALREADY_TRACKING, 27

 CONNECTION_FAILED, 27

 DEVICE_MISSING, 27

 ErrorType, 26

 INVALID_CALIBRATION_MODE, 27

 INVALID_FRAMERATE_MODE, 27

 NOT_CALIBRATED, 27

 NOT_CONNECTED, 27

 NOT_INITED, 26

 NOT_TRACKING, 27

 SCREEN_FAILURE, 27

 SCREEN_NOT_FOUND, 27

 UNKNOWN_ERROR, 26

 VERSION_MISMATCH, 26

ErrorType

 EException, 26

eyelogic, 17

 CONNECTION_CLOSED, 18

 DEVICE_CONNECTED, 18

 DEVICE_DISCONNECTED, 18

 DeviceEventType, 17

 SCREEN_CHANGED, 18

 TRACKING_STOPPED, 18

eyePositionLeft

 GazeSample, 28

eyePositionRight

 GazeSample, 28

GazeSample, 27

 eyePositionLeft, 28

 eyePositionRight, 28

getActiveScreen

 ELCsApi, 23

getAvailableScreens

 ELCsApi, 23

getDeviceConfig

 ELCsApi, 24

INVALID_CALIBRATION_MODE

 EException, 27

INVALID_FRAMERATE_MODE

 EException, 27

NOT_CALIBRATED

 EException, 27

NOT_CONNECTED

 EException, 27

NOT_INITED
 EException, 26

NOT_TRACKING
 EException, 27

Point2d, 28

Point3d, 29

requestServerList
 ELCsApi, 24

requestTracking
 ELCsApi, 24

SCREEN_CHANGED
 eyelogic, 18

SCREEN_FAILURE
 EException, 27

SCREEN_NOT_FOUND
 EException, 27

setActiveScreen
 ELCsApi, 25

TRACKING_STOPPED
 eyelogic, 18

UNKNOWN_ERROR
 EException, 26

unrequestTracking
 ELCsApi, 25

validate
 ELCsApi, 25

VERSION_MISMATCH
 EException, 26
