



EyeLogic SDK

v 1.1.15

March 2025



---

<b>1 EyeLogic SDK Documentation (C++)</b>	<b>1</b>
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	7
1.4.1 License Agreement and Warranty for SDK	7
1.5 About EyeLogic	8
1.5.1 Contact and Support	8
<b>2 Namespace Index</b>	<b>9</b>
2.1 Namespace List	9
<b>3 Class Index</b>	<b>11</b>
3.1 Class List	11
<b>4 File Index</b>	<b>13</b>
4.1 File List	13
<b>5 Namespace Documentation</b>	<b>15</b>
5.1 elapi Namespace Reference	15
5.1.1 Detailed Description	15
<b>6 Class Documentation</b>	<b>17</b>
6.1 ELApi::DeviceConfig Struct Reference	17
6.1.1 Detailed Description	17
6.2 ELApi::DeviceGeometry Struct Reference	18
6.2.1 Detailed Description	18
6.3 ELApi Class Reference	18
6.3.1 Detailed Description	20
6.3.2 Member Enumeration Documentation	20
6.3.2.1 DeviceEvent	20

---

6.3.2.2 ReturnCalibrate . . . . .	21
6.3.2.3 ReturnConnect . . . . .	21
6.3.2.4 ReturnNextData . . . . .	21
6.3.2.5 ReturnSetActiveScreen . . . . .	22
6.3.2.6 ReturnStart . . . . .	22
6.3.2.7 ReturnStreamEyeImages . . . . .	22
6.3.2.8 ReturnValidate . . . . .	23
6.3.3 Constructor & Destructor Documentation . . . . .	23
6.3.3.1 ELApi() . . . . .	23
6.3.4 Member Function Documentation . . . . .	23
6.3.4.1 calibrate() . . . . .	24
6.3.4.2 connect() . . . . .	24
6.3.4.3 connectRemote() . . . . .	24
6.3.4.4 getAvailableScreens() . . . . .	24
6.3.4.5 getNextDeviceEvent() . . . . .	26
6.3.4.6 getNextEyeImage() . . . . .	26
6.3.4.7 getNextGazeSample() . . . . .	27
6.3.4.8 registerDeviceEventListener() . . . . .	28
6.3.4.9 registerEyeImageListener() . . . . .	28
6.3.4.10 registerGazeEventListener() . . . . .	28
6.3.4.11 registerGazeSampleListener() . . . . .	29
6.3.4.12 requestServerList() . . . . .	29
6.3.4.13 requestTracking() . . . . .	29
6.3.4.14 setActiveScreen() . . . . .	30
6.3.4.15 streamEyeImages() . . . . .	30
6.3.4.16 unrequestTracking() . . . . .	31
6.3.4.17 validate() . . . . .	31
6.4 ELApi::ELDeviceEventCallback Class Reference . . . . .	31
6.4.1 Detailed Description . . . . .	31
6.4.2 Member Function Documentation . . . . .	31
6.4.2.1 onDeviceEvent() . . . . .	31
6.5 ELEyeImage Struct Reference . . . . .	32
6.5.1 Detailed Description . . . . .	32
6.5.2 Member Data Documentation . . . . .	32
6.5.2.1 SIZE . . . . .	32
6.6 ELApi::ELEyeImageCallback Class Reference . . . . .	33
6.6.1 Detailed Description . . . . .	33
6.6.2 Member Function Documentation . . . . .	33
6.6.2.1 onEyeImage() . . . . .	33
6.7 ELFixationStart Struct Reference . . . . .	33
6.7.1 Detailed Description . . . . .	34
6.8 ELFixationStop Struct Reference . . . . .	34

6.8.1 Detailed Description . . . . .	34
6.9 ELApi::ELGazeEventCallback Class Reference . . . . .	34
6.9.1 Detailed Description . . . . .	35
6.9.2 Member Function Documentation . . . . .	35
6.9.2.1 onFixationStart() . . . . .	35
6.9.2.2 onFixationStop() . . . . .	35
6.10 ELGazeSample Struct Reference . . . . .	35
6.10.1 Detailed Description . . . . .	36
6.10.2 Member Data Documentation . . . . .	36
6.10.2.1 eyePositionLeftX . . . . .	37
6.10.2.2 eyePositionLeftY . . . . .	37
6.10.2.3 eyePositionLeftZ . . . . .	37
6.10.2.4 eyePositionRightX . . . . .	37
6.10.2.5 eyePositionRightY . . . . .	38
6.10.2.6 eyePositionRightZ . . . . .	38
6.11 ELApi::ELGazeSampleCallback Class Reference . . . . .	38
6.11.1 Detailed Description . . . . .	38
6.11.2 Member Function Documentation . . . . .	38
6.11.2.1 onGazeSample() . . . . .	38
6.12 ELApi::ELValidationPointResult Struct Reference . . . . .	39
6.12.1 Detailed Description . . . . .	39
6.13 ELApi::ELValidationResult Struct Reference . . . . .	39
6.13.1 Detailed Description . . . . .	40
6.14 ELApi::ScreenConfig Struct Reference . . . . .	40
6.14.1 Detailed Description . . . . .	40
6.15 ELApi::ServerInfo Struct Reference . . . . .	40
6.15.1 Detailed Description . . . . .	40
<b>7 File Documentation . . . . .</b>	<b>41</b>
7.1 ELApi.h File Reference . . . . .	41
7.1.1 Detailed Description . . . . .	42
7.2 ELEyelImage.h File Reference . . . . .	42
7.2.1 Detailed Description . . . . .	42
7.3 ELGazeEvent.h File Reference . . . . .	42
7.3.1 Detailed Description . . . . .	43
7.4 ELGazeSample.h File Reference . . . . .	43
7.4.1 Detailed Description . . . . .	43
<b>Index . . . . .</b>	<b>45</b>



# 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. It is built for Microsoft Windows (32 bit or 64 bit) only. The included sample projects are written for Microsoft Visual Studio 2017 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 cpp folder, the content should be:

- bin - contains the binary DLLs to link against
- example - contains the sample code
- include - contains the include header files for compilation

### 1.2.3 Getting Started with the Sample Code

In the directory, into which you unpacked the SDK EyeLogicSDK, navigate to the sub-directory `cpp/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:

```
1>----- Build started: Project: DemoClient, Configuration: Debug x64 -----
2>----- Build started: Project: DualPC, Configuration: Debug x64 -----
3>----- Build started: Project: Validation, Configuration: Debug x64 -----
1> main_democlient.cpp
2> dualpc_democlient.cpp
3> validation_democlient.cpp
1> DemoClient.vcxproj -> cpp\example\x64\Debug\DemoClient.exe
2> DualPC.vcxproj -> cpp\example\x64\Debug\DualPC.exe
3> Validation.vcxproj -> cpp\example\x64\Debug\Validation.exe
1> Copy dll dependencies for execution
1>      1 File copied.
2> Copy dll dependencies for execution
2>      1 File copied.
3> Copy dll dependencies for execution
3>      1 File copied.
===== Build: 3 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```



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:

- Under "C/C++", set "Additional Include Directories" to the location of `<Location of your EyeLogic_SDK>\cpp\include`.
- Under "Linker", set "Additional Library Dependencies" to `<Location of your EyeLogic_SDK>\cpp\bin`.
- Under "Linker -> Input", add `ELApi.lib` to "Additional Dependencies" (for Win32-Applications, use `ELApi32.lib`).

### 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. It adds

```
#include "elapi/ELApi.h"
```

in order to find all necessary definitions of the EyeLogic API.

Gaze samples and events are populated by asynchronous callbacks. They are defined further below by deriving from the interfaces `elapi::ELApi::ELGazeSampleCallback` and `elapi::ELApi::ELEventCallback`.

Events are fired whenever something changes in the external state, such as a new eye tracking device being connected. The definition of the event receiver is:

```
class EventReceiver : public elapi::ELApi::ELEventCallback
{
public:
    onEvent( elapi::ELApi::Event event ) override { ... }
};
```

and the definition of the gaze sample receiver is:

```
class GazeSampleReceiver : public elapi::ELApi::ELGazeSampleCallback
{
    void
    onGazeSample( const elapi::ELGazeSample& gazeSample ) override { ... }
}
```

The example code simply prints incoming gaze samples and events to the console.

The `main( )` method is where the application implements its control flow. It consists of the following lines of code:

```
DeviceListener deviceListener;
elapi::ELApi    api( "Demo Client" );
auto eventReceiver = std::make_unique< EventReceiver >( api );
api.registerEventListener( eventReceiver.get( ) );
```

This constructs a new instance of the `ELApi` class. The instantiation will automatically initialize the library and it will also be automatically deinitialized when object `api` goes out of scope. The call to `registerEventListener` registers your own instance of the event callback with the EyeLogic API. From now on all incoming events will call the `onEvent( )` method from the code above.

```
const auto retConnect = api.connect( );
```

Connects to the EyeLogic server. Check the return code to see if the connection was established successfully.

```
api.getActiveScreen( screenConfig );
```

---

and

```
api.getDeviceConfig( deviceConfig );
```

are called in order to obtain information about the active screen and the connected eye tracking device.

```
const auto retStart = 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 10 seconds and then closes the connection:

```
api.disconnect( );  
api.registerGazeSampleListener( nullptr );  
api.registerEventListener( nullptr );
```

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 guaranties 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 are 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](mailto: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:

<b>elapi</b>	Namespace for C++ API calls . . . . .	<b>15</b>
--------------	---------------------------------------	-----------





## Chapter 3

# Class Index

### 3.1 Class List

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

<b>ELApi::DeviceConfig</b>	
Device configuration . . . . .	17
<b>ELApi::DeviceGeometry</b>	
Geometric position of the device related to the active monitor . . . . .	18
<b>ELApi</b>	
Main class for communication with the EyeLogic server . . . . .	18
<b>ELApi::ELDeviceEventCallback</b>	
Callback interface for events related to the eye tracker . . . . .	31
<b>ELEyeImage</b>	
Image of the eyes captured by the device . . . . .	32
<b>ELApi::ELEyeImageCallback</b>	
Callback interface for EyeImages . . . . .	33
<b>ELFixationStart</b>	
Information about a fixation start . . . . .	33
<b>ELFixationStop</b>	
Information about a fixation end . . . . .	34
<b>ELApi::ELGazeEventCallback</b>	
Callback interface for gaze related events . . . . .	34
<b>ELGazeSample</b>	
All information about the state of the eyes at a specific time . . . . .	35
<b>ELApi::ELGazeSampleCallback</b>	
Callback interface for gaze samples . . . . .	38
<b>ELApi::ELValidationPointResult</b>	
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] . . . . .	39
<b>ELApi::ELValidationResult</b>	
ValidationResult contains one ValidationPointResult struct per validation stimulus point of the performed validation . . . . .	39
<b>ELApi::ScreenConfig</b>	
Screen configuration . . . . .	40
<b>ELApi::ServerInfo</b>	
Connection information for an EyeLogic server . . . . .	40



## Chapter 4

# File Index

### 4.1 File List

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

<b>ELApi.h</b>	The file contains the C++ prototype declaration for all functions which are necessary to control the EyeLogic software from an API client . . . . .	<b>41</b>
<b>ELEyeImage.h</b>	The file specifies the C++ container for an eye image . . . . .	<b>42</b>
<b>ELGazeEvent.h</b>	The file specifies the C++ container for a gaze event . . . . .	<b>42</b>
<b>ELGazeSample.h</b>	The file specifies the C++ container for a gaze sample . . . . .	<b>43</b>



## Chapter 5

# Namespace Documentation

### 5.1 elapi Namespace Reference

namespace for C++ API calls

#### Classes

- class **ELApi**  
*main class for communication with the EyeLogic server*
- struct **ELEyeImage**  
*contains an image of the eyes captured by the device*
- struct **ELFixationStart**  
*information about a fixation start*
- struct **ELFixationStop**  
*information about a fixation end*
- struct **ELGazeSample**  
*contains all information about the state of the eyes at a specific time*

#### Variables

- `EL_EXPORT` const double **ELInvalidValue**  
*marker for an invalid double value*

#### 5.1.1 Detailed Description

namespace for C++ API calls



## Chapter 6

# Class Documentation

### 6.1 ELApi::DeviceConfig Struct Reference

Device configuration.

```
#include "ELApi.h"
```

#### Public Attributes

- uint64\_t **deviceSerial**  
*serial number of the device as unsigned 64-bit int for a verbose format, print it as 8-digit hex number*
- char **deviceName** [32]  
*name of the device, 0-terminated string*
- char **brandedName** [64]  
*name of the license owner, 0-terminated string*
- bool **isDemoDevice**  
*whether the device is for DEMO use only, not for public sale*
- int32\_t **numFrameRates**  
*number of available framerates*
- uint8\_t **frameRates** [16]  
*array of available framerates [Hz], use only the entries frameRates[0] ... frameRates[numFrameRates-1]*
- int32\_t **numCalibrationMethods**  
*number of available calibration methods*
- uint8\_t **calibrationMethods** [16]  
*array of available calibration methods [number of calibration points], use only the entries calibrationMethods[0] ... calibrationMethods[numCalibrationMethods-1]*

#### 6.1.1 Detailed Description

Device configuration.

## 6.2 ELApi::DeviceGeometry Struct Reference

Geometric position of the device related to the active monitor.

```
#include "ELApi.h"
```

### 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*

### 6.2.1 Detailed Description

Geometric position of the device related to the active monitor.

## 6.3 ELApi Class Reference

main class for communication with the EyeLogic server

```
#include "ELApi.h"
```

### Classes

- struct **DeviceConfig**  
*Device configuration.*
- struct **DeviceGeometry**  
*Geometric position of the device related to the active monitor.*
- class **ELDeviceEventCallback**  
*Callback interface for events related to the eye tracker.*
- class **ELEyeImageCallback**  
*Callback interface for EyeImages.*
- class **ELGazeEventCallback**  
*Callback interface for gaze related events.*
- class **ELGazeSampleCallback**  
*Callback interface for gaze samples.*
- struct **ELValidationPointResult**  
*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].*
- struct **ELValidationResult**  
*ValidationResult contains one ValidationPointResult struct per validation stimulus point of the performed validation.*
- struct **ScreenConfig**  
*Screen configuration.*
- struct **ServerInfo**  
*connection information for an EyeLogic server*



## Public Types

- enum **DeviceEvent** {  
**SCREEN\_CHANGED**, **CONNECTION\_CLOSED**, **DEVICE\_CONNECTED**, **DEVICE\_DISCONNECTED**,  
**TRACKING\_STOPPED** }  
*event reated to the state of the eye tracker*
- enum **ReturnConnect** { **SUCCESS**, **FAILURE**, **VERSION\_MISMATCH** }  
*return values of `connect()`*
- enum **ReturnSetActiveScreen** { **SUCCESS**, **NOT\_FOUND**, **FAILURE** }  
*return values of `setActiveScreen()`*
- enum **ReturnStreamEyeImages** { **SUCCESS**, **NOT\_CONNECTED**, **REMOTE\_CONNECTION**, **FAILURE** }  
*Return values of the `streamEyeImages()` function.*
- enum **ReturnNextData** { **SUCCESS**, **TIMEOUT**, **CONNECTION\_CLOSED** }  
*Return values of the `getNextEvent/getNextGazeSample` functions.*
- enum **ReturnStart** {  
**SUCCESS**, **NOT\_CONNECTED**, **DEVICE\_MISSING**, **INVALID\_FRAMERATE\_MODE**,  
**ALREADY\_RUNNING\_DIFFERENT\_FRAMERATE**, **FAILURE** }  
*return values of `requestTracking()`*
- enum **ReturnCalibrate** {  
**SUCCESS**, **NOT\_CONNECTED**, **NOT\_TRACKING**, **INVALID\_CALIBRATION\_MODE**,  
**ALREADY\_BUSY**, **FAILURE** }  
*return values of `calibrate()`*
- enum **ReturnValidate** {  
**SUCCESS**, **NOT\_CONNECTED**, **NOT\_TRACKING**, **NOT\_CALIBRATED**,  
**ALREADY\_BUSY**, **FAILURE** }  
*return values of `validate()`*

## Public Member Functions

- **EL\_EXPORT** **STD**CALL **ELApi** (const char \*clientName)  
*constructor*
- **EL\_EXPORT** **STD**CALL **~ELApi** ()  
*destructor*
- **ELApi** (const **ELApi** &)=delete
- **ELApi** & **operator=** (const **ELApi** &)=delete
- **ELApi** (**ELApi** &&)=delete
- **ELApi** & **operator=** (**ELApi** &&)=delete
- **EL\_EXPORT** void **STD**CALL **registerDeviceEventListener** (**ELDeviceEventCallback** \*callback)  
*Registers the event listener. An existing listener will be overwritten.*
- **EL\_EXPORT** void **STD**CALL **registerGazeEventListener** (**ELGazeEventCallback** \*callback)  
*Registers the event listener. An existing listener will be overwritten.*
- **EL\_EXPORT** void **STD**CALL **registerGazeSampleListener** (**ELGazeSampleCallback** \*callback)  
*Registers the gaze sample listener. An existing listener will be overwritten.*
- **EL\_EXPORT** void **STD**CALL **registerEyeImageListener** (**ELEyeImageCallback** \*callback)  
*Registers the eye image listener. An existing listener will be overwritten.*
- **EL\_EXPORT** **ReturnConnect** **STD**CALL **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,*
- **EL\_EXPORT** **ReturnConnect** **STD**CALL **connectRemote** (**ServerInfo** server)  
*initialize connection to a remote server (method is blocking until connection established)*
- **EL\_EXPORT** int32\_t **STD**CALL **requestServerList** (int32\_t blockingDurationMS, **ServerInfo** \*serverList, int32\_t serverListLength)

- Ping all running EyeLogic servers in the local network and wait some time for their response.*
- EL\_EXPORT void STDCALL **disconnect** ()  
*closes connection to the server*
  - EL\_EXPORT bool STDCALL **isConnected** () const  
*whether a connection to the server is established*
  - EL\_EXPORT void STDCALL **getActiveScreen** (ScreenConfig &screenConfig) const  
*obtain configuration of active screen*
  - EL\_EXPORT int32\_t STDCALL **getAvailableScreens** (ScreenConfig \*screenConfig, int32\_t numScreenConfigs) const  
*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.*
  - EL\_EXPORT ReturnSetActiveScreen STDCALL **setActiveScreen** (const char \*screenID, DeviceGeometry deviceGeometry)  
*Make a screen connected to this machine to the active screen.*
  - EL\_EXPORT void STDCALL **getDeviceConfig** (DeviceConfig &deviceConfig) const  
*obtain configuration of active device*
  - EL\_EXPORT ReturnStreamEyeImages STDCALL **streamEyeImages** (bool enable)  
*Enabled/disables eye image stream. If enabled, eye images are received from eye image listeners,.*
  - EL\_EXPORT ReturnNextData STDCALL **getNextDeviceEvent** (DeviceEvent &event, int32\_t timeoutMillis)  
*Obtains the next unread event or blocks until a new event occurs or the given timeout is reached.*
  - EL\_EXPORT ReturnNextData STDCALL **getNextGazeSample** (ELGazeSample &gazeSample, int32\_t timeoutMillis)  
*Obtains the next unread gazeSample or blocks until a new GazeSample is received or the given timeout is reached.*
  - EL\_EXPORT ReturnNextData STDCALL **getNextEyeImage** (ELEyeImage &eyeImage, int32\_t timeoutMillis)  
*Obtains the next unread eye image or blocks until a new eye image is received or the given timeout is reached.*
  - EL\_EXPORT ReturnStart STDCALL **requestTracking** (int32\_t frameRateModelInd)  
*request tracking*
  - EL\_EXPORT void STDCALL **unrequestTracking** ()  
*unrequest tracking*
  - EL\_EXPORT ReturnCalibrate STDCALL **calibrate** (int32\_t calibrationModelInd)  
*perform calibration (method is blocking until calibration finished)*
  - EL\_EXPORT void STDCALL **abortCalibValidation** ()  
*abort a running calibration / validation*
  - EL\_EXPORT ReturnValidate STDCALL **validate** (ELValidationResult &validationResult)  
*perform validation (method is blocking until validation finished)*

### 6.3.1 Detailed Description

main class for communication with the EyeLogic server

### 6.3.2 Member Enumeration Documentation

#### 6.3.2.1 DeviceEvent

```
enum DeviceEvent [strong]
```

event related to the state of the eye tracker

**Enumerator**

SCREEN_CHANGED	active screen or resolution has changed
CONNECTION_CLOSED	connection to server has closed
DEVICE_CONNECTED	a new device has connected
DEVICE_DISCONNECTED	actual device has disconnected
TRACKING_STOPPED	tracking has stopped

**6.3.2.2 ReturnCalibrate**

```
enum ReturnCalibrate [strong]
```

return values of `calibrate()`

**Enumerator**

SUCCESS	start calibration successful
NOT_CONNECTED	cannot calibrate: not connected to the server
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_BUSY	cannot start calibration: calibration or validation is already in progress
FAILURE	calibration failure

**6.3.2.3 ReturnConnect**

```
enum ReturnConnect [strong]
```

return values of `connect()`

**Enumerator**

SUCCESS	connection successfully established
FAILURE	connection could not be established: the server can not be found or is not responding
VERSION_MISMATCH	connection could not be established: API is build on a newer version than the server. Update the EyeLogicServer to the newest version.

**6.3.2.4 ReturnNextData**

```
enum ReturnNextData [strong]
```

Return values of the `getNextEvent/getNextGazeSample` functions.

---

## Enumerator

SUCCESS	new event or new GazeSample received
TIMEOUT	timeout reached, no new event/GazeSample received
CONNECTION_CLOSED	connection to server closed, no new event/GazeSample received

**6.3.2.5 ReturnSetActiveScreen**

```
enum ReturnSetActiveScreen [strong]
```

return values of `setActiveScreen()`

## Enumerator

SUCCESS	active screen was set
NOT_FOUND	specified screen name was not found as a name of an available monitor
FAILURE	active screen could not be changed

**6.3.2.6 ReturnStart**

```
enum ReturnStart [strong]
```

return values of `requestTracking()`

## Enumerator

SUCCESS	start tracking successful
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_RUNNING_DIFFERENT_FRAMERATE	tracking already ongoing, but frame rate mode is different
FAILURE	some general failure occurred

**6.3.2.7 ReturnStreamEyeImages**

```
enum ReturnStreamEyeImages [strong]
```

Return values of the `streamEyeImages()` function.

---

## Enumerator

SUCCESS	setting streaming of eye images was successful
NOT_CONNECTED	failed, not connected to the server
REMOTE_CONNECTION	cannot stream eye images when connection to the server is a remote connection
FAILURE	failure when trying to set eye image stream

## 6.3.2.8 ReturnValidate

```
enum ReturnValidate [strong]
```

return values of `validate()`

## Enumerator

SUCCESS	start validation successful
NOT_CONNECTED	cannot validate: not connected to the server
NOT_TRACKING	cannot validate: no device found or tracking not started
NOT_CALIBRATED	cannot start validation: validation mode is invalid or not supported
ALREADY_BUSY	cannot start validation: calibration or validation is already in progress
FAILURE	validation failure

## 6.3.3 Constructor &amp; Destructor Documentation

## 6.3.3.1 ELApi()

```
EL_EXPORT STDCALL ELApi (
    const char * clientName )
```

constructor

## Parameters

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

## 6.3.4 Member Function Documentation

#### 6.3.4.1 `calibrate()`

```
EL_EXPORT ReturnCalibrate STDCALL calibrate (
    int32_t calibrationModeInd )
```

perform calibration (method is blocking until calibration finished)

##### Parameters

<i>calibrationModeInd</i>	index of the requested calibration method (0 ... #calibrationMethods-1)
---------------------------	---

#### 6.3.4.2 `connect()`

```
EL_EXPORT ReturnConnect STDCALL 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,

##### See also

`connectRemote()`.

#### 6.3.4.3 `connectRemote()`

```
EL_EXPORT ReturnConnect STDCALL connectRemote (
    ServerInfo server )
```

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

#### 6.3.4.4 `getAvailableScreens()`

```
EL_EXPORT int32_t STDCALL getAvailableScreens (
    ScreenConfig * screenConfig,
    int32_t numScreenConfigs ) const
```

---

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

**6.3.4.5 getNextDeviceEvent()**

```
EL_EXPORT ReturnNextData STDCALL getNextDeviceEvent (
    DeviceEvent & event,
    int32_t timeoutMillis )
```

Obtains the next unread event or blocks until a new event occurs or the given timeout is reached.

The last incoming event is buffered internally and can be obtained by calling this method in a consecutive order. If there is no new event, the method blocks until an event occurs or the given timeout is reached. The method returns SUCCESS if and only if a new event is provided which was not returned before. Therefore, by checking the return value, you can assure to not handle any event twice.

If you want to catch events in a loop, be careful to not wait too long between the calls to this method. Otherwise, you may miss events. If you want to be 100% sure to not miss any event, consider to use the [ELDeviceEventCallback](#) mechanism.

**See also**

[registerEventListener](#)

**Parameters**

<i>event</i>	If this method returns SUCCESS, this data structure is filled with the new (yet unhandled) event. In all other cases, this data structure is filled with the event which was returned last.
<i>timeoutMillis</i>	duration in milliseconds, method returns at the latest after this time. May be 0 if the method should return immediatly.

**Returns**

whether an event was received (SUCCESS) or the method terminated without a new event

**6.3.4.6 getNextEyeImage()**

```
EL_EXPORT ReturnNextData STDCALL getNextEyeImage (
    ELEyeImage & eyeImage,
    int32_t timeoutMillis )
```



Obtains the next unread eye image or blocks until a new eye image is received or the given timeout is reached.

The last incoming eye image is buffered internally and can be obtained by calling this method in a consecutive order. If there is no new eye image, the method blocks until an eye image is received or the given timeout is reached. The method returns SUCCESS if and only if a new eye image is provided which was not returned before. Therefore, by checking the return value, you can assure to not handle any eye image twice.

#### Parameters

<i>eyeImage</i>	If this method returns SUCCESS, this data structure is filled with the new (yet unhandled) eye image. In all other cases, this data structure is filled with the eye image which was returned last.
<i>timeoutMillis</i>	duration in milliseconds, method returns at the latest after this time. May be 0 if the method should return immediatly.

#### Returns

whether an eye iage was received (SUCCESS)

#### 6.3.4.7 getNextGazeSample()

```
EL_EXPORT ReturnNextData STDCALL getNextGazeSample (
    ELGazeSample & gazeSample,
    int32_t timeoutMillis )
```

Obtains the next unread gazeSample or blocks until a new GazeSample is received or the given timeout is reached.

The last incoming GazeSample is buffered internally and can be obtained by calling this method in a consecutive order. If there is no new GazeSample, the method blocks until a GazeSample arrives or the given timeout is reached. The method returns SUCCESS if and only if a new GazeSample is provided which was not returned before. Therefore, by checking the return value, you can assure to not handle any GazeSample twice.

If you want to catch GazeSamples in a loop, be careful to not wait too long between the calls to this method (at least once per frame). Otherwise, you may miss GazeSamples. If you want to be 100% sure to not miss any GazeSample, consider to use the [ELGazeSampleCallback](#) mechanism.

#### See also

[registerGazeSampleListener](#)

#### Parameters

<i>gazeSample</i>	If this method returns SUCCESS, this data structure is filled with the new (yet unhandled) GazeSample. In all other cases, this data structure is filled with the GazeSample which was returned last.
<i>timeoutMillis</i>	duration in milliseconds, method returns at the latest after this time. May be 0 if the method should return immediatly.

**Returns**

whether a GazeSample was received (SUCCESS) or the method terminated without a new GazeSample

**6.3.4.8 registerDeviceEventListener()**

```
EL_EXPORT void STDCALL registerDeviceEventListener (
    ELDeviceEventCallback * callback )
```

Registers the event listener. An existing listener will be overwritten.

**Parameters**

<i>callback</i>	this instance will be notified of all events published by the <a href="#">ELApi</a> . If null, the current callback is removed/unregistered. Ensure that the listener is unregistered before its destruction.
-----------------	---

**6.3.4.9 registerEyeImageListener()**

```
EL_EXPORT void STDCALL registerEyeImageListener (
    ELEyeImageCallback * callback )
```

Registers the eye image listener. An existing listener will be overwritten.

Note: Eye images must be enabled to receive those callbacks

**See also**

[streamEyeImages\(\)](#)

**Parameters**

<i>callback</i>	this instance will be notified of all eye images published by the <a href="#">ELApi</a> . If null, the current callback is removed/unregistered. Ensure that the listener is unregistered before its destruction.
-----------------	---

**6.3.4.10 registerGazeEventListener()**

```
EL_EXPORT void STDCALL registerGazeEventListener (
    ELGazeEventCallback * callback )
```

Registers the event listener. An existing listener will be overwritten.

---

## Parameters

<i>callback</i>	this instance will be notified of all events published by the <b>ELApi</b> . If null, the current callback is removed/unregistered. Ensure that the listener is unregistered before its destruction.
-----------------	--

**6.3.4.11 registerGazeSampleListener()**

```
EL_EXPORT void STDCALL registerGazeSampleListener (
    ELGazeSampleCallback * callback )
```

Registers the gaze sample listener. An existing listener will be overwritten.

## Parameters

<i>callback</i>	this instance will be notified of all gaze samples published by the <b>ELApi</b> . If null, the current callback is removed/unregistered. Ensure that the listener is unregistered before its destruction.
-----------------	--

**6.3.4.12 requestServerList()**

```
EL_EXPORT int32_t STDCALL requestServerList (
    int32_t blockingDurationMS,
    ServerInfo * serverList,
    int32_t serverListLength )
```

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 after this time, or if 'serverListLength' many servers responded.
<i>serverList</i>	pre-allocated array of length 'serverListLength'. Will be filled with responding EyeLogic servers.
<i>serverListLength</i>	Length of pre-allocated serverList array

## Returns

number of entries, written to the server list

**6.3.4.13 requestTracking()**

```
EL_EXPORT ReturnStart STDCALL requestTracking (
    int32_t frameRateModeInd )
```

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 (0 ... #frameRateModes-1)
--------------------------	--

#### 6.3.4.14 setActiveScreen()

```
EL_EXPORT ReturnSetActiveScreen STDCALL setActiveScreen (
    const char * screenID,
    DeviceGeometry deviceGeometry )
```

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

#### 6.3.4.15 streamEyeImages()

```
EL_EXPORT ReturnStreamEyeImages STDCALL streamEyeImages (
    bool enable )
```

Enabled/disables eye image stream. If enabled, eye images are received from eye image listeners,.

#### See also

[registerEyeImageListener\(\)](#) and

[getNextEyeImage\(\)](#). Note, that enabling eye images can lead to noticable CPU load and a loss of gaze samples. Always disable it before running your experiment. Eye images can not be received via remote connections.

### 6.3.4.16 unrequestTracking()

```
EL_EXPORT void STDCALL unrequestTracking ( )
```

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.

### 6.3.4.17 validate()

```
EL_EXPORT ReturnValidate STDCALL validate (
    ELValidationResult & validationResult )
```

perform validation (method is blocking until validation finished)

#### Parameters

<i>validationResult</i>	upon <b>ReturnValidate::SUCCESS</b> this struct will be filled with the validation results - may contain ELInvalidValues. Contains all ELInvalidValues for all other return values.
-------------------------	---

## 6.4 ELApi::ELDeviceEventCallback Class Reference

Callback interface for events related to the eye tracker.

```
#include "ELApi.h"
```

### Public Member Functions

- virtual void STDCALL **onDeviceEvent** (ELApi::DeviceEvent event)=0  
*Callback function for new events.*

### 6.4.1 Detailed Description

Callback interface for events related to the eye tracker.

### 6.4.2 Member Function Documentation

#### 6.4.2.1 onDeviceEvent()

```
virtual void STDCALL onDeviceEvent (
    ELApi::DeviceEvent event ) [pure virtual]
```

Callback function for new events.

---

## Parameters

<i>event</i>	The occurred event
--------------	--------------------

## 6.5 ELEyeImage Struct Reference

contains an image of the eyes captured by the device

```
#include "ELEyeImage.h"
```

### Public Attributes

- `uint8_t data [SIZE]`  
*image buffer, stores all pixels as RGB value (3 bytes per pixel)*

### Static Public Attributes

- `static const int32_t WIDTH = 300`  
*width of the image in pixels*
- `static const int32_t HEIGHT = 90`  
*height of the image in pixels*
- `static const int32_t SIZE = WIDTH * HEIGHT * 3`  
*size of the image buffer*

### 6.5.1 Detailed Description

contains an image of the eyes captured by the device

### 6.5.2 Member Data Documentation

#### 6.5.2.1 SIZE

```
const int32_t SIZE = WIDTH * HEIGHT * 3 [static]
```

size of the image buffer

#### See also

`data` in byte

## 6.6 ELApi::ELEyeImageCallback Class Reference

Callback interface for EyeImages.

```
#include "ELApi.h"
```

### Public Member Functions

- virtual void STDCALL **onEyeImage** (const **ELEyeImage** &eyeImage)=0

*Callback function for new eye images. Note that this callback is only invoked for direct connections, not for remote connections.*

#### 6.6.1 Detailed Description

Callback interface for EyeImages.

#### 6.6.2 Member Function Documentation

##### 6.6.2.1 onEyeImage()

```
virtual void STDCALL onEyeImage (
    const ELEyeImage & eyeImage ) [pure virtual]
```

Callback function for new eye images. Note that this callback is only invoked for direct connections, not for remote connections.

##### Parameters

eye	image incoming eye image
-----	--------------------------

## 6.7 ELFixationStart Struct Reference

information about a fixation start

```
#include "ELGazeEvent.h"
```

### Public Attributes

- int64\_t **timestampMicroSec**  
*timepoint when the fixation started in microseconds after EPOCH*
- int32\_t **index**

- index of the corresponding GazeSample at which the fixation started*
- double **porX**  
*X coordinate of binocular point of regard on the stimulus plane at when the fixation started.*
- double **porY**  
*Y coordinate of binocular point of regard on the stimulus plane at when the fixation started.*

### 6.7.1 Detailed Description

information about a fixation start

## 6.8 ELFixationStop Struct Reference

information about a fixation end

```
#include "ELGazeEvent.h"
```

### Public Attributes

- int64\_t **timestampMicroSec**  
*timepoint when the fixation ended in microseconds after EPOCH*
- int64\_t **timestampStartMicroSec**  
*timepoint when the fixation started in microseconds after EPOCH*
- int32\_t **index**  
*index of the corresponding GazeSample at which the fixation ended*
- int32\_t **indexStart**  
*index of the corresponding GazeSample at which the fixation started*
- double **porX**  
*X coordinate of binocular point of regard on the stimulus plane of the overal fixation (average over the whole fixation period)*
- double **porY**  
*Y coordinate of binocular point of regard on the stimulus plane of the overal fixation (average over the whole fixation period)*

### 6.8.1 Detailed Description

information about a fixation end

## 6.9 ELApi::ELGazeEventCallback Class Reference

Callback interface for gaze related events.

```
#include "ELApi.h"
```

---



## Public Member Functions

- virtual void STDCALL **onFixationStart** (**ELFixationStart** fixation)=0  
*Callback function for a detected fixation start.*
- virtual void STDCALL **onFixationStop** (**ELFixationStop** fixation)=0  
*Callback function for a detected fixation stop.*

### 6.9.1 Detailed Description

Callback interface for gaze related events.

### 6.9.2 Member Function Documentation

#### 6.9.2.1 onFixationStart()

```
virtual void STDCALL onFixationStart (  
    ELFixationStart fixation ) [pure virtual]
```

Callback function for a detected fixation start.

##### Parameters

<i>fixation</i>	info about the fixation
-----------------	-------------------------

#### 6.9.2.2 onFixationStop()

```
virtual void STDCALL onFixationStop (  
    ELFixationStop fixation ) [pure virtual]
```

Callback function for a detected fixation stop.

##### Parameters

<i>fixation</i>	info about the fixation
-----------------	-------------------------

## 6.10 ELGazeSample Struct Reference

contains all information about the state of the eyes at a specific time

```
#include "ELGazeSample.h"
```

---

## Public Attributes

- `int64_t timestampMicroSec`  
*timepoint when data was acquired in microseconds after EPOCH*
- `int32_t index`  
*increasing GazeSample index*
- `double porRawX`  
*X coordinate of binocular point of regard on the stimulus plane, check `porRawX != InvalidValue` before using it.*
- `double porRawY`  
*Y coordinate of binocular point of regard on the stimulus plane, check `porRawX != InvalidValue` also before using `porRawY`.*
- `double porFilteredX`  
*X coordinate of filtered binocular point of regard on the stimulus plane, check `porFilteredX != InvalidValue` before using it.*
- `double porFilteredY`  
*Y coordinate of filtered binocular point of regard on the stimulus plane, also check `porFilteredX != InvalidValue` before using `porFilteredY`.*
- `double porLeftX`  
*X coordinate of monocular point of regard of the left eye, check `porLeftX != InvalidValue` before using it.*
- `double porLeftY`  
*Y coordinate of monocular point of regard of the left eye, also check `porLeftX != InvalidValue` before using `porLeftY`.*
- `double eyePositionLeftX`  
*position of the left eye in device coordinates, unit is mm*
- `double eyePositionLeftY`  
*position of the left eye in device coordinates, unit is mm*
- `double eyePositionLeftZ`  
*position of the left eye in device coordinates, unit is mm*
- `double pupilRadiusLeft`  
*radius of the left pupil in mm or `InvalidValue` if eye was not found*
- `double porRightX`  
*X coordinate of monocular point of regard of the right eye, check `porRightX != InvalidValue` before using it.*
- `double porRightY`  
*Y coordinate of monocular point of regard of the right eye, also check `porRightX != InvalidValue` before using `porRightY`.*
- `double eyePositionRightX`  
*position of the right eye in device coordinates, unit is mm:*
- `double eyePositionRightY`  
*position of the right eye in device coordinates, unit is mm:*
- `double eyePositionRightZ`  
*position of the right eye in device coordinates, unit is mm:*
- `double pupilRadiusRight`  
*radius of the right pupil in mm or `InvalidValue` if eye was not found*

### 6.10.1 Detailed Description

contains all information about the state of the eyes at a specific time

### 6.10.2 Member Data Documentation

### 6.10.2.1 eyePositionLeftX

```
double eyePositionLeftX
```

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

check eyePositionLeftX != InvalidValue before using it

### 6.10.2.2 eyePositionLeftY

```
double eyePositionLeftY
```

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

- (0, 0, 0) is in the center of the device
- y-coordinate: positive towards the top of the screen

check eyePositionLeftX != InvalidValue before using eyePositionLeftY

### 6.10.2.3 eyePositionLeftZ

```
double eyePositionLeftZ
```

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

- (0, 0, 0) is in the center of the device
- z-coordinate: distance in front of the screen

check eyePositionLeftX != InvalidValue before using eyePositionLeftZ

### 6.10.2.4 eyePositionRightX

```
double eyePositionRightX
```

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

check eyePositionRightX != InvalidValue before using it

---

### 6.10.2.5 eyePositionRightY

```
double eyePositionRightY
```

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

- (0, 0, 0) is in the center of the device
- y-coordinate: positive towards the top of the screen

check eyePositionRightX != InvalidValue before using eyePositionRightY

### 6.10.2.6 eyePositionRightZ

```
double eyePositionRightZ
```

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

- (0, 0, 0) is in the center of the device
- z-coordinate: distance in front of the screen

check eyePositionRightX != InvalidValue before using eyePositionRightZ

## 6.11 ELApi::ELGazeSampleCallback Class Reference

Callback interface for gaze samples.

```
#include "ELApi.h"
```

### Public Member Functions

- virtual void STDCALL **onGazeSample** (const **ELGazeSample** &gazeSample)=0  
*Callback function for new gaze samples.*

#### 6.11.1 Detailed Description

Callback interface for gaze samples.

#### 6.11.2 Member Function Documentation

##### 6.11.2.1 onGazeSample()

```
virtual void STDCALL onGazeSample (
    const ELGazeSample & gazeSample ) [pure virtual]
```

Callback function for new gaze samples.

---

## Parameters

<i>gazeSample</i>	incoming gaze sample
-------------------	----------------------

## 6.12 ELApi::ELValidationPointResult Struct 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].

```
#include "ELApi.h"
```

### Public Attributes

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

### 6.12.1 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: All data fields may be ELInvalidValue. All pairs validationPointPxX/-Y, meanDeviationLeftDeg/-Px and meanDeviationRightDeg/-Px are always either both valid or both ELInvalidValue.

## 6.13 ELApi::ELValidationResult Struct Reference

ValidationResult contains one ValidationPointResult struct per validation stimulus point of the performed validation.

```
#include "ELApi.h"
```

## Public Attributes

- `ELValidationPointResult pointsData` [4]

### 6.13.1 Detailed Description

ValidationResult contains one ValidationResult struct per validation stimulus point of the performed validation.

## 6.14 ELApi::ScreenConfig Struct Reference

Screen configuration.

```
#include "ELApi.h"
```

## Public Attributes

- `bool localMachine`  
*whether the screen is connected to the this machine*
- `char id` [32]  
*identifier name of the screen (0-terminated string)*
- `char name` [32]  
*descriptive name of the screen (0-terminated string)*
- `int32_t resolutionX`  
*raw screen X resolution [px]*
- `int32_t resolutionY`  
*raw screen Y resolution [px]*
- `double physicalSizeX_mm`  
*horizontal physical dimension of the screen [mm]*
- `double physicalSizeY_mm`  
*vertical physical dimension of the screen [mm]*

### 6.14.1 Detailed Description

Screen configuration.

## 6.15 ELApi::ServerInfo Struct Reference

connection information for an EyeLogic server

```
#include "ELApi.h"
```

## Public Attributes

- `char ip` [16]  
*IP address of server as 0-terminated string.*
- `uint16_t port`  
*port of server*

### 6.15.1 Detailed Description

connection information for an EyeLogic server

---

## Chapter 7

# File Documentation

### 7.1 ELApi.h File Reference

The file contains the C++ prototype declaration for all functions which are necessary to control the EyeLogic software from an API client.

```
#include "ELGazeSample.h"
#include "ELEyeImage.h"
#include "ELGazeEvent.h"
#include <memory>
```

#### Classes

- class **ELApi**  
*main class for communication with the EyeLogic server*
- class **ELApi::ELDeviceEventCallback**  
*Callback interface for events related to the eye tracker.*
- class **ELApi::ELGazeEventCallback**  
*Callback interface for gaze related events.*
- class **ELApi::ELGazeSampleCallback**  
*Callback interface for gaze samples.*
- class **ELApi::ELEyeImageCallback**  
*Callback interface for EyeImages.*
- struct **ELApi::ServerInfo**  
*connection information for an EyeLogic server*
- struct **ELApi::ScreenConfig**  
*Screen configuration.*
- struct **ELApi::DeviceGeometry**  
*Geometric position of the device related to the active monitor.*
- struct **ELApi::DeviceConfig**  
*Device configuration.*
- struct **ELApi::ELValidationPointResult**  
*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].*
- struct **ELApi::ELValidationResult**  
*ValidationResult contains one ValidationPointResult struct per validation stimulus point of the performed validation.*

## Namespaces

- **elapi**  
*namespace for C++ API calls*

### 7.1.1 Detailed Description

The file contains the C++ prototype declaration for all functions which are necessary to control the EyeLogic software from an API client.

## 7.2 ELEyelImage.h File Reference

The file specifies the C++ container for an eye image.

```
#include "ELExports.hpp"  
#include <cinttypes>
```

### Classes

- struct **ELEyelImage**  
*contains an image of the eyes captured by the device*

## Namespaces

- **elapi**  
*namespace for C++ API calls*

### 7.2.1 Detailed Description

The file specifies the C++ container for an eye image.

## 7.3 ELGazeEvent.h File Reference

The file specifies the C++ container for a gaze event.

```
#include "ELExports.hpp"  
#include <cinttypes>
```

### Classes

- struct **ELFixationStart**  
*information about a fixation start*
  - struct **ELFixationStop**  
*information about a fixation end*
-



## Namespaces

- `elapi`  
*namespace for C++ API calls*

### 7.3.1 Detailed Description

The file specifies the C++ container for a gaze event.

## 7.4 ELGazeSample.h File Reference

The file specifies the C++ container for a gaze sample.

```
#include "ELExports.hpp"  
#include <cinttypes>
```

## Classes

- struct `ELGazeSample`  
*contains all information about the state of the eyes at a specific time*

## Namespaces

- `elapi`  
*namespace for C++ API calls*

## Variables

- `EL_EXPORT` const double `ELInvalidValue`  
*marker for an invalid double value*

### 7.4.1 Detailed Description

The file specifies the C++ container for a gaze sample.

---



# Index

- ALREADY\_BUSY
  - ELApi, 21, 23
- ALREADY\_RUNNING\_DIFFERENT\_FRAMERATE
  - ELApi, 22
- calibrate
  - ELApi, 23
- connect
  - ELApi, 24
- CONNECTION\_CLOSED
  - ELApi, 21, 22
- connectRemote
  - ELApi, 24
- DEVICE\_CONNECTED
  - ELApi, 21
- DEVICE\_DISCONNECTED
  - ELApi, 21
- DEVICE\_MISSING
  - ELApi, 22
- DeviceEvent
  - ELApi, 20
- ELApi, 18
  - ALREADY\_BUSY, 21, 23
  - ALREADY\_RUNNING\_DIFFERENT\_FRAMERATE, 22
  - calibrate, 23
  - connect, 24
  - CONNECTION\_CLOSED, 21, 22
  - connectRemote, 24
  - DEVICE\_CONNECTED, 21
  - DEVICE\_DISCONNECTED, 21
  - DEVICE\_MISSING, 22
  - DeviceEvent, 20
  - ELApi, 23
  - FAILURE, 21–23
  - getAvailableScreens, 24
  - getNextDeviceEvent, 26
  - getNextEyeImage, 26
  - getNextGazeSample, 27
  - INVALID\_CALIBRATION\_MODE, 21
  - INVALID\_FRAMERATE\_MODE, 22
  - NOT\_CALIBRATED, 23
  - NOT\_CONNECTED, 21–23
  - NOT\_FOUND, 22
  - NOT\_TRACKING, 21, 23
  - registerDeviceEventListener, 28
  - registerEyeImageListener, 28
  - registerGazeEventListener, 28
  - registerGazeSampleListener, 29
  - REMOTE\_CONNECTION, 23
  - requestServerList, 29
  - requestTracking, 29
  - ReturnCalibrate, 21
  - ReturnConnect, 21
  - ReturnNextData, 21
  - ReturnSetActiveScreen, 22
  - ReturnStart, 22
  - ReturnStreamEyeImages, 22
  - ReturnValidate, 23
  - SCREEN\_CHANGED, 21
  - setActiveScreen, 30
  - streamEyeImages, 30
  - SUCCESS, 21–23
  - TIMEOUT, 22
  - TRACKING\_STOPPED, 21
  - unrequestTracking, 30
  - validate, 31
  - VERSION\_MISMATCH, 21
- elapi, 15
- ELApi.h, 41
- ELApi::DeviceConfig, 17
- ELApi::DeviceGeometry, 18
- ELApi::ELDeviceEventCallback, 31
  - onDeviceEvent, 31
- ELApi::ELEyeImageCallback, 33
  - onEyeImage, 33
- ELApi::ELGazeEventCallback, 34
  - onFixationStart, 35
  - onFixationStop, 35
- ELApi::ELGazeSampleCallback, 38
  - onGazeSample, 38
- ELApi::ELValidationPointResult, 39
- ELApi::ELValidationResult, 39
- ELApi::ScreenConfig, 40
- ELApi::ServerInfo, 40
- ELEyeImage, 32
  - SIZE, 32
- ELEyeImage.h, 42
- ELFixationStart, 33
- ELFixationStop, 34
- ELGazeEvent.h, 42
- ELGazeSample, 35
  - eyePositionLeftX, 36
  - eyePositionLeftY, 37
  - eyePositionLeftZ, 37
  - eyePositionRightX, 37
  - eyePositionRightY, 37

- eyePositionRightZ, 38
  - ELGazeSample.h, 43
  - eyePositionLeftX
    - ELGazeSample, 36
  - eyePositionLeftY
    - ELGazeSample, 37
  - eyePositionLeftZ
    - ELGazeSample, 37
  - eyePositionRightX
    - ELGazeSample, 37
  - eyePositionRightY
    - ELGazeSample, 37
  - eyePositionRightZ
    - ELGazeSample, 38
  - FAILURE
    - ELApi, 21–23
  - getAvailableScreens
    - ELApi, 24
  - getNextDeviceEvent
    - ELApi, 26
  - getNextEyelImage
    - ELApi, 26
  - getNextGazeSample
    - ELApi, 27
  - INVALID\_CALIBRATION\_MODE
    - ELApi, 21
  - INVALID\_FRAMERATE\_MODE
    - ELApi, 22
  - NOT\_CALIBRATED
    - ELApi, 23
  - NOT\_CONNECTED
    - ELApi, 21–23
  - NOT\_FOUND
    - ELApi, 22
  - NOT\_TRACKING
    - ELApi, 21, 23
  - onDeviceEvent
    - ELApi::ELDeviceEventCallback, 31
  - onEyelImage
    - ELApi::ELEyelImageCallback, 33
  - onFixationStart
    - ELApi::ELGazeEventCallback, 35
  - onFixationStop
    - ELApi::ELGazeEventCallback, 35
  - onGazeSample
    - ELApi::ELGazeSampleCallback, 38
  - registerDeviceEventListener
    - ELApi, 28
  - registerEyelImageListener
    - ELApi, 28
  - registerGazeEventListener
    - ELApi, 28
  - registerGazeSampleListener
    - ELApi, 29
  - REMOTE\_CONNECTION
    - ELApi, 23
  - requestServerList
    - ELApi, 29
  - requestTracking
    - ELApi, 29
  - ReturnCalibrate
    - ELApi, 21
  - ReturnConnect
    - ELApi, 21
  - ReturnNextData
    - ELApi, 21
  - ReturnSetActiveScreen
    - ELApi, 22
  - ReturnStart
    - ELApi, 22
  - ReturnStreamEyelImages
    - ELApi, 22
  - ReturnValidate
    - ELApi, 23
  - SCREEN\_CHANGED
    - ELApi, 21
  - setActiveScreen
    - ELApi, 30
  - SIZE
    - ELEyelImage, 32
  - streamEyelImages
    - ELApi, 30
  - SUCCESS
    - ELApi, 21–23
  - TIMEOUT
    - ELApi, 22
  - TRACKING\_STOPPED
    - ELApi, 21
  - unrequestTracking
    - ELApi, 30
  - validate
    - ELApi, 31
  - VERSION\_MISMATCH
    - ELApi, 21
-