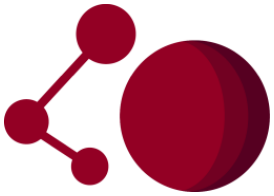

EyeLogic SDK

v 1.1.9

September 2023

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 which use an EyeLogic eye tracking device. It offers the possibility to connect with your device via an application programming interface (API) from any custom application. The EyeLogic SDK is available for the programming languages C++, C#, C, and Python. It is also usable with any other programming language that is capable of importing dynamic link libraries (DLLs), e.g. Visual Basic or Matlab.

For each directly supported language, there is a short and simple sample program to help you get started with the development of your first EyeLogic application.

This guide describes the use of 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

For the system requirements of the EyeLogic Server and an installation guide, please refer to the Server's documentation.

The SDK has no additional requirements. It is built for Microsoft Windows only (32 bit or 64 bit). The included sample projects are written for Microsoft Visual Studio 2017 or newer. Any other compilers are not yet supported.

1.2 Installation and Getting Started

1.2.1 Download Software

In order to use an EyeLogic eye tracking device from within your application, you need the EyeLogic Server and the EyeLogic SDK. Check the download-page to get the latest release of both packages: https://www.eyelogicsolutions.com/downloads/

1.2.1.1 Compatibility

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

On the other hand, updating the SDK and API-DLLs is not always neccessary. Since you as a programmer would have to recompile your application with every SDK-update, we designed the SDK such that the server is able to communicate with older API versions. Therefore, when shipping your application, just add the EyeLogic API DLLs of the actual version to your package. It is compatible with servers of the actual and newer releases.

See Shipping your Application for a tutorial on how to ship your application.

1.2.2 Install EyeLogic SDK on Windows

The EyeLogic SDK does not need to be installed. It ships as .zip file which just needs to be extracted to some directory on your hard disk. Be sure, that you have user-rights to that directory, e.g. any directory inside C:\Program Files or similar is problematic, since it requires admin rights to access those files every time you start your program. It is recommended to use a user-local 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 an output similarly to:

```
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 is an EyeLogic icon in the windows tray bar.

In the left part of the editor, there is a list of all projects / democlients. The active one is marked in bold (DemoClient). You might make any other demo client active (e.g. DualPC or Validation) by right-click on the desired name in the list and set it as the Startup Project.

Press F5 to compile and run the application.

Note that your firewall might 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 the windows defender, just click "accept").

If you reached this point, you have properly set up your EyeLogic SDK. You may now start with the development of you 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 neccessary 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 which can be used by your application to set up a connection to the EyeLogic Server, start tracking and receive eye tracking data.

The server is designed to run permanently on your computer as a background process. While not actively tracking the server requires an insignificant portion of your computer's resources. Once an EyeLogic eye tracking device is plugged in, the server application detects it automatically and allows the user to set it up via the servers' configuration dialog (see the server icon in the windows tray bar). If for any reason the server background process is not running (the tray icon is missing), you may start the server manually via the windows start menu.

The API is a set of .dll files which can be used by any custom program (called the `user application`). Using those DLLs the user application can establish a connection to the (running) server. Note that it the EyeLogic Server may run on the same computer than the user application, or they may run on different PCs. See Dual PC Setup for how to set up the setting with running the server and the user application on different computers.

1.3.2 Set Up a Project for your Application

For an easy start to develop a new application it is recommended to 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. Starting from this sample code, you can easily modify and extend the code to suit your customized experiment.

Alternatively you can start a new Visual Studio project from scratch. In that case be sure the compiler and linker are able to find the EyeLogic include and binary files. Therefore, apply 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 characterized 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 `GazeSample`s to the user application, see also GazeSamples.
5. **perform calibration:** Request a calibration. The screen will show a calibration point animated to be moving across the screen. The user must fixate on this point until the calibration screen diappears. The system is calibrated and ready to use once this process is completed sucessfully.
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 will be transmitted via `asyncroneous callbacks`. The application has to register it's own implementations of those callback functions with the API (see Example Program for an example implementation).

Note that you need to calibrate in order to obtain valid gaze samples (see GazeSamples). All gaze samples which are reported before the system is calibrated contain no valid eye data.

1.3.4 Dual PC Setup

The Dual-PC setup is a special setting 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 who constrols the eye tracking device and a participant who has to perform a task. The participant uses a different PC (showing the experiment) than the operator (who can control the eye tracker via the EyeLogic Server software).

The computer of the operator (called Operator PC) needs to have the EyeLogic driver software (the EyeLogic Server) installed and running. The eye tracker is physically mounted to a screen which is connected to the computer of the participant (called Experiment PC). The USB cable of the eye tracker is plugged into the USB port of the Operator PC!

Now, the operator can use the server to detect the eye tracking device. On the Experiment PC, any custom application which shows an experiment to the participant, can use the EyeLogic API to connect to the server remotely. In order to do that, the application should use the API calls:

1. `requestServerList()` 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. `connectRemote()` to conntect to a specific server from that list
3. `setActiveScreen()` 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 has to allow remote connections in order to be found. You can enable that in the settings of the server window.

If connected successfully, the client can operate as usual as if it would be connected to a local server. See the demo application "dualpc" 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 neccessary definitions of the EyeLogic API.

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

Events are invoked whenever something on the external state changes, e.g. a new eyetracking device is plugged in. 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.

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

```
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 instanciation will automatically initialize the library and it will also be automatically deinitialized when object api goes out of scope. The call of `registerEventListener` registers the own instance of the event callback to the EyeLogic API. From now on, any incoming events will invoke the `onEvent( )` method from the code above.

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

Connects to the EyeLogic server. Check for the return value in order to find out whether 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 begin sample processing. The parameter 0 specifies the frame rate mode. If your device is capable of multiple frame rate modes (60Hz, 120Hz or 250Hz), you can also enter 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 usually is the highest available speed mode of your system.

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

Performs a calibration. This method blocks until the calibration ends - i.e. completed or aborted. The parameter 0 denotes 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 GazeSamples

GazeSamples are the most essential data which is generated by the eye tracker. The eye tracker delivers one GazeSample per frame. Each sample contains information on the time of measurement, the position of the eyes, the pupil radius and the point where the user looks at on some 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 may run on different computers. The EyeLogic functionality will only work on computers which have the EyeLogic Server installed. The installed server needs to at least be of the same version as the shipped API DLLs (a newer server version is permissible).

Beside the relevant files of your application, you need to ship the content of the bin/ folder of your language (typically including some .dll files). Place the content of the bin/ folder inside 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 no 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 Eye↩ Logic 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 I 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 you 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 ere foreseeable and whether or not EyeLogic was advices of the possibility of such damages. Without limiting the generality of the foregoing, EyeLogic's aggregate liability to you (whether under contract, tort, statue 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

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

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 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
- 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 ELEventCallback

  Callback interface for events related to the eye tracker.
- class ELEyeImageCallback

  Callback interface for EyeImages.
- 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, validation↩ PointPxY) [px].
- struct ELValidationResult

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

  Screen configuration.
- struct ServerInfo

  connection information for an EyeLogic server

Public Types

- enum Event {
  SCREEN_CHANGED, CONNECTION_CLOSED, DEVICE_CONNECTED, DEVICE_DISCONNECTED,
  TRACKING_STOPPED }

  eye tracking event
- 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 STDCALL ELApi (const char ∗clientName)

  constructor
- EL_EXPORT STDCALL ∼ELApi ()

  destructor
- **ELApi** (const ELApi &)=delete
- ELApi & **operator=** (const ELApi &)=delete
- **ELApi** (ELApi &&)=delete
- ELApi & **operator=** (ELApi &&)=delete
- EL_EXPORT void STDCALL registerEventListener (ELEventCallback ∗callback)

  Registers the event listener. An existing listener will be overwritten.
- EL_EXPORT void STDCALL registerGazeSampleListener (ELGazeSampleCallback ∗callback)

  Registers the gaze sample listener. An existing listener will be overwritten.
- EL_EXPORT void STDCALL registerEyeImageListener (ELEyeImageCallback ∗callback)

  Registers the eye image listener. An existing listener will be overwritten.
- 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,
- EL_EXPORT ReturnConnect STDCALL connectRemote (ServerInfo server)

  initialize connection to a remote server (method is blocking until connection established)
- 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.
- 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 getNextEvent (Event &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 frameRateModeInd)

  request tracking

- EL_EXPORT void STDCALL unrequestTracking ()

  unrequest tracking

- EL_EXPORT ReturnCalibrate STDCALL calibrate (int32_t calibrationModeInd)

  perform calibration (method is blocking until calibration finished)

- EL_EXPORT void STDCALL abortCalibValidation ()

  abort a running calibration

- 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 Event

```
enum Event  [strong]
```

eye tracking event

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 successully 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 & Destructor Documentation

6.3.3.1 ELApi()

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

constructor

Parameters

clientName	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

calibrationModeInd	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

server	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

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

Returns

number of filled screen configurations. will be <= numScreenConfigs

6.3.4.5 getNextEvent()

```
EL_EXPORT ReturnNextData STDCALL getNextEvent (
            Event & 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 ELEventCallback mechanism.

See also

registerEventListener

Parameters

event	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.
timeoutMillis	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

eyeImage	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.
timeoutMillis	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

gazeSample	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.
timeoutMillis	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 registerEventListener()

```
EL_EXPORT void STDCALL registerEventListener (
            ELEventCallback * callback )
```

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

Parameters

callback	this instance will be notified of all events published by the ELApi. 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

callback	this instance will be notified of all eye images published by the ELApi. If null, the current callback is removed/unregistered. Ensure that the listener is unregistered before its destruction.

6.3.4.10 registerGazeSampleListener()

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

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

Parameters

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

6.3.4.11 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

blockingDurationMS	waiting duration in milliseconds. Method returns after this time, or if 'serverListLength' many servers responded.
serverList	pre-allocated array of length 'serverListLength'. Will be filled with responding EyeLogic servers.
serverListLength	Lenght of pre-allocated serverList array

Returns

number of entries, written to the server list

6.3.4.12 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

frameRateModeInd	index of the requested frame rate mode (0 ... #frameRateModes-1)

6.3.4.13 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

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

Returns

success/error code

6.3.4.14 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.

6.3.4.15 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.

6.3.4.16 validate()

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

perform validation (method is blocking until validation finished)

Parameters

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

6.4 ELApi::ELEventCallback Class Reference

Callback interface for events related to the eye tracker.

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

Public Member Functions

- virtual void STDCALL onEvent (ELApi::Event 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 onEvent()

```
virtual void STDCALL onEvent (
            ELApi::Event event )  [pure virtual]
```

Callback function for new events.

Parameters

event	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 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

6.7.1 Detailed Description

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

6.7.2 Member Data Documentation

6.7.2.1 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

6.7.2.2 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

6.7.2.3 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

6.7.2.4 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

6.7.2.5 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

6.7.2.6 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

6.8 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.

6.8.1 Detailed Description

Callback interface for gaze samples.

6.8.2 Member Function Documentation

6.8.2.1 onGazeSample()

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

Callback function for new gaze samples.

Parameters

gazeSample	incoming gaze sample

6.9 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.

6.9.1 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 mean↩ DeviationRightDeg-/Px are always either both valid or both ELInvalidValue.

6.10 ELApi::ELValidationResult Struct Reference

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

`#include "ELApi.h"`

Public Attributes

- ELValidationPointResult **pointsData** [4]

6.10.1 Detailed Description

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

6.11 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]

6.11.1 Detailed Description

Screen configuration.

6.12 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

6.12.1 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 neccessary to control the EyeLogic software from an API client.

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

Classes

- class ELApi

  main class for communication with the EyeLogic server
- class ELApi::ELEventCallback

  Callback interface for events related to the eye tracker.
- 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, validation↩ PointPxY) [px].
- struct ELApi::ELValidationResult

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

Namespaces

- elapi

  namespace for C++ API calls

7.1.1 Detailed Description

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

7.2 ELEyeImage.h File Reference

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

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

Classes

- struct ELEyeImage

  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 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

7.3.1 Detailed Description

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

Index