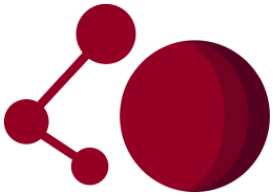

EyeLogic SDK

v 1.1.9

September 2023

Chapter 1

EyeLogic SDK Documentation (Python)

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 Python and gives a step-by-step introduction on how to start with your own Python 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 python folder, the content should be:

- eyelogic - contains the EyeLogic package which can be included in your Python script
- democlient.py - a sample script which demonstrates the use of the EyeLogic python API

1.2.3 Getting Started with the Sample Code

In the directory, into which you unpacked the SDK `EyeLogicSDK`, navigate to the sub-directory `python`. Open the one of the .py files with your favorite python development environment.

If you have your python interpreter in the windows PATH, then you may start the demo application by just double-clicking the .py file, e.g. democlient_main_sample_polling.py. Alternatively, open a console, change the actual directory to `EyeLogicSDK\python` and enter the following line:

```
python democlient_main_sample_polling.py
```

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.

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\python 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 python project from scratch. In that case be sure that your development environment is able to find the path for the EyeLogic python module (which is `<Location of your EyeLogic_SDK>\python`.

1.3.3 Control Flow between API and server

The usual control flow between the custom application/API and 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 Python example program is explained in some detail.

The file starts with an include section. It adds

```
from eyelogic.ELApi import *
```

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

The next relevant part is the definition of the callback functions.:

```
@SampleCallback
def sampleCallback(sample: POINTER(ELGazeSample))


@EventCallback
def eventCallback(event: ELEvent)
```

These are the callback functions which are invoked from the EyeLogic software whenever an event occurs. Those functions are defined in the following lines. The example code simply prints the event to the console, but here you may write your custom implementation.

In the __main__ section, the application implements its control flow. It consists of the following code lines:

```
api = ELApi("Demo Client")
api.registerEventCallback(eventCallback)
```

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 `registerEventCallback` registers the own instance of the event callback to the EyeLogic API. From now on, any incoming events will invoke the `eventCallback( )` method from the code above.

```
resultConnect = api.connect()
```

Connects to the EyeLogic server. Check for the return value in order to find out whether the connection was established successfully.

```
screenConfig = api.getActiveScreen()
```

and

```
deviceConfig = api.getDeviceConfig()
```

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

```
resultTracking = 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 passed to the callback onDeviceConnected( ) whereas the first frame rate mode (0) is the default mode, which usually is the highest available speed mode of your system.

```
resultCalibrate = 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 api.getDeviceConfig( ).

The example program waits for 10 seconds and then closes the connection:

```
api.disconnect()
api.registerGazeSampleCallback(None)
api.registerEventCallback(None)
```

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 eyelogic/ subfolder (from <SDK path>/python with all its content. The python interpreter must be able to locate this folder in order to locate the eyelogic module on your destination machine. You may place the eyelogic/ folder inside the working directory of your application and ship them altogether.

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 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 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, Eye↩ Logic 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 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 ELApi Namespace Reference

Classes

- class ELApi

  main class for communication with the EyeLogic server
- class ELEvent

  Events coming from the eye tracker.
- class ELEyeImage

  contains an eye image at a specific time
- class ELGazeSample

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

Functions

- def **check_bool** (result, func, args)

Variables

- GazeSampleCallback = CFUNCTYPE(None, POINTER(ELGazeSample))

  callback function type, new gaze samples
- EventCallback = CFUNCTYPE(None, c_int32)

  callback function type, event occurred
- EyeImageCallback = CFUNCTYPE(None, POINTER(ELEyeImage))

  callback function type, new eye image
- **libname** = os.path.join("x64", "ELCApi")
- **baseDir** = os.path.dirname(os.path.abspath(__file__))
- **libnameGlobal** = os.path.join(baseDir, libname + ".dll")
- **kernel32** = WinDLL('kernel32', use_last_error=True)
- **errcheck**
- **restype**
- **argtypes**
- **c_libH** = kernel32.LoadLibraryExW(libnameGlobal, None, 0x00000008)
- **c_lib** = WinDLL(libname, handle=c_libH)
- ELInvalidValue = c_double.in_dll(c_lib, "ELCInvalidValue").value

  marker for an invalid double value

5.1.1 Detailed Description

This module contains the python prototype declaration for all functions which are neccessary to control the EyeLogic software from an API client.

Chapter 6

Class Documentation

6.1 ELApi.DeviceConfig Class Reference

configuration of the eye tracker

Public Member Functions

- def __init__ (self, deviceSerial)
  constructor

Public Attributes

- deviceSerial
  serial number
- frameRates
  list of supported frame rates
- calibrationMethods
  list of supported calibration methods (number of shown points)

6.1.1 Detailed Description

configuration of the eye tracker

6.1.2 Constructor & Destructor Documentation

6.1.2.1 __init__()

```
def __init__ (
              self,
              deviceSerial )
```

constructor

Parameters

deviceSerial	serial number of the device

6.2 ELApi.DeviceGeometry Class Reference

geometric position of the device related to the active monitor

Public Member Functions

- def __init__ (self, mmBelowScreen, mmTrackerInFrontOfScreen)
  constructor

Public Attributes

- mmBelowScreen
  distance of eye tracker below the bottom line of the screen [mm]
- mmTrackerInFrontOfScreen
  distance of front panel of the eye tracker in front of the screen[mm]

6.2.1 Detailed Description

geometric position of the device related to the active monitor

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
def __init__ (
              self,
              mmBelowScreen,
              mmTrackerInFrontOfScreen )
```

constructor

Parameters

mmBelowScreen	distance of eye tracker below the bottom line of the screen [mm]
mmTrackerInFrontOfScreen	distance of front panel of the eye tracker in front of the screen[mm]

6.3 ELApi Class Reference

main class for communication with the EyeLogic server

Classes

- class DeviceConfig
  configuration of the eye tracker
- class DeviceGeometry
  geometric position of the device related to the active monitor
- class ReturnCalibrate
  return values of calibrate( )
- class ReturnConnect
  return values of connect( )
- class ReturnNextData
  return values of getNextEvent( ), getNextGazeSample( ) and getNextEyeImage( )
- class ReturnSetActiveScreen
  return values of setActiveScreen( )
- class ReturnStart
  return values of requestTracking( )
- class ReturnStreamEyeImages
  return values of streamEyeImages( )
- class ReturnValidate
  return values of validate( )
- class ScreenConfig
  configuration of the stimulus screen
- class ServerInfo
  connection information for an EyeLogic server
- class ValidationPointResult
  ValidationPointResult.
- class ValidationResult
  ValidationResult.

Public Member Functions

- def __init__ (self, str clientName)
  constructor
- def __del__ (self)
  destructor
- def registerGazeSampleCallback (self, GazeSampleCallback sampleCallback)
  registers sample callback listener
- def registerEyeImageCallback (self, EyeImageCallback eyeImageCallback)
  registers eye image callback listener
- def registerEventCallback (self, EventCallback eventCallback)
  registers event callback listener
- ReturnConnect connect (self)
  initialize connection to the server (method is blocking until connection established).
- ReturnConnect connectRemote (self, ServerInfo server)
  initialize connection to a remote server (method is blocking until connection established)

- [ServerInfo] requestServerList (self, c_int32 blockingDurationMS, c_int32 maxNumServer)

  Ping all running EyeLogic servers in the local network and wait some time for their response.
- def disconnect (self)

  closes connection to the server
- bool isConnected (self)

  whether a connection to the server is established
- ScreenConfig getActiveScreen (self)

  get stimulus screen configuration
- [ScreenConfig] getAvailableScreens (self)

  Get a list of screens connected to the local machine.
- ReturnSetActiveScreen setActiveScreen (self, str id, DeviceGeometry deviceGeometry)

  Make a screen connected to this machine to the active screen.
- DeviceConfig getDeviceConfig (self)

  get configuration of actual eye tracker device
- ReturnStreamEyeImages streamEyeImages (self, c_bool enable)

  Enabled/disables eye image stream.
- (ReturnNextData, ELEvent) getNextEvent (self, c_int timeoutMillis)

  Obtains the next unread event or blocks until a new event occurs or the given timeout is reached.
- (ReturnNextData, ELGazeSample) getNextGazeSample (self, c_int timeoutMillis)

  Obtains the next unread gazeSample or blocks until a new GazeSample is received or the given timeout is reached.
- (ReturnNextData, ELEyeImage) getNextEyeImage (self, c_int timeoutMillis)

  Obtains the next unread eye image or blocks until a new eye image is received or the given timeout is reached.
- ReturnStart requestTracking (self, c_int frameRateModeInd)

  request tracking
- def unrequestTracking (self)

  unrequest tracking
- def calibrate (self, c_int calibrationModeInd)

  perform calibration (method is blocking until calibration finished)
- (ReturnValidate, ValidationResult) validate (self)

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

6.3.1 Detailed Description

main class for communication with the EyeLogic server

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __init__()

```
def __init__ (
              self,
             str clientName )
```

constructor

Parameters

clientName	string identifier of the client (shown in the server tool window), may be null

6.3.3 Member Function Documentation

6.3.3.1 calibrate()

```
def calibrate (
              self,
             c_int calibrationModeInd )
```

perform calibration (method is blocking until calibration finished)

Returns

success state

6.3.3.2 connect()

```
ReturnConnect connect (
              self )
```

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( ).

Returns

success state

6.3.3.3 connectRemote()

```
ReturnConnect connectRemote (
              self,
             ServerInfo server )
```

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

Parameters

server	Server to connect to

Returns

success state

See also

acquireServerList( ) to obtain IP address and port of a remote server

6.3.3.4 getActiveScreen()

```
ScreenConfig getActiveScreen (
              self )
```

get stimulus screen configuration

Returns

screen configuration

6.3.3.5 getAvailableScreens()

```
[ScreenConfig] getAvailableScreens (
              self )
```

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.

Returns

list of screen configurations

6.3.3.6 getDeviceConfig()

```
DeviceConfig getDeviceConfig (
              self )
```

get configuration of actual eye tracker device

Returns

device configuration

6.3.3.7 getNextEvent()

```
(ReturnNextData, ELEvent) getNextEvent (
              self,
             c_int 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

timeoutMillis	duration in milliseconds, method returns at the latest after this time. May be 0 if the method should return immediatly.

Returns

first: new (yet unhandled) event. second: whether an event was received (SUCCESS) or the method terminated without a new event

6.3.3.8 getNextEyeImage()

```
(ReturnNextData, ELEyeImage) getNextEyeImage (
              self,
             c_int  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.

If you want to catch EyeImages 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 EyeImages. If you want to be 100% sure to not miss any EyeImages, consider to use the ELEyeImagesCallback mechanism.

See also

registerEyeImagesListener

Parameters

timeoutMillis	duration in milliseconds, method returns at the latest after this time. May be 0 if the method should return immediatly.

Returns

first: new (yet unhandled) EyeImages. second: whether an event was received (SUCCESS)

6.3.3.9 getNextGazeSample()

```
(ReturnNextData, ELGazeSample) getNextGazeSample (
              self,
             c_int  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

timeoutMillis	duration in milliseconds, method returns at the latest after this time. May be 0 if the method should return immediatly.

Returns

first: new (yet unhandled) GazeSample. second: whether an event was received (SUCCESS) or the method terminated without a new GazeSample

6.3.3.10 registerEventCallback()

```
def registerEventCallback (
              self,
            EventCallback eventCallback )
```

registers event callback listener

Parameters

eventCallback	this callback function is called on eye tracking events, may be null

6.3.3.11 registerEyeImageCallback()

```
def registerEyeImageCallback (
              self,
             EyeImageCallback eyeImageCallback )
```

registers eye image callback listener

Parameters

eyeImageCallback	this callback function is called on new eye images, may be null

6.3.3.12 registerGazeSampleCallback()

```
def registerGazeSampleCallback (
              self,
             GazeSampleCallback sampleCallback )
```

registers sample callback listener

Parameters

sampleCallback	this callback function is called on new gaze samples, may be null

6.3.3.13 requestServerList()

```
[ServerInfo] requestServerList (
              self,
             c_int32 blockingDurationMS,
             c_int32 maxNumServer )
```

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.
maxNumServer	maximum number of server to be waited for

Returns

List of responding EyeLogic servers

6.3.3.14 requestTracking()

```
ReturnStart requestTracking (
              self,
             c_int 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)

Returns

success state

6.3.3.15 setActiveScreen()

```
ReturnSetActiveScreen setActiveScreen (
              self,
             str id,
             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

id	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

6.3.3.16 streamEyeImages()

```
ReturnStreamEyeImages streamEyeImages (
              self,
             c_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.

6.3.3.17 unrequestTracking()

```
def unrequestTracking (
              self )
```

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.

6.3.3.18 validate()

```
(ReturnValidate, ValidationResult) validate (
              self )
```

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

Returns

whether was completed successfully (SUCCESS) or error value and an instance of ValidationResult. Upon SUCCESS ValidationResult.pointsData will contain each stimulus point's validation data, empty list otherwise.

6.4 ELEvent Class Reference

Events coming from the eye tracker.

Static Public Attributes

- int SCREEN_CHANGED = 0
  screen or resolution has changed
- int CONNECTION_CLOSED = 1
  connection to server closed
- int DEVICE_CONNECTED = 2
  a new eye tracker has connected
- int DEVICE_DISCONNECTED = 3
  the actual eye tracker has disconnected
- int TRACKING_STOPPED = 4
  tracking stopped

6.4.1 Detailed Description

Events coming from the eye tracker.

6.5 ELEyeImage Class Reference

contains an eye image at a specific time

6.5.1 Detailed Description

contains an eye image at a specific time

6.6 ELGazeSample Class Reference

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

6.6.1 Detailed Description

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

Available members:

- **timestampMicroSec**: timepoint when data was acquired in microseconds after EPOCH
- **index**: increasing GazeSample index
- **porRawX**: X coordinate of binocular point of regard on the stimulus plane, check porRawX != InvalidValue before using it.
- **porRawY**: Y coordinate of binocular point of regard on the stimulus plane, check porRawX != InvalidValue also before using porRawY.
- **porFilteredX**: X coordinate of filtered binocular point of regard on the stimulus plane, check porFilteredX != InvalidValue before using it.

- **porFilteredY**: Y coordinate of filtered binocular point of regard on the stimulus plane, also check porFilteredX != InvalidValue before using porFilteredY.
- **porLeftX**: X coordinate of monocular point of regard of the left eye, check porLeftX != InvalidValue before using it.
- **porLeftY**: Y coordinate of monocular point of regard of the left eye, also check porLeftX != InvalidValue before using porLeftY.
- **eyePositionLeftX**: position of the left eye in device coordinates, unit is mm More...
- **eyePositionLeftY**: position of the left eye in device coordinates, unit is mm More...
- **eyePositionLeftZ**: position of the left eye in device coordinates, unit is mm More...
- **pupilRadiusLeft**: radius of the left pupil in mm
- **porRightX**: X coordinate of monocular point of regard of the right eye, check porRightX != InvalidValue before using it.
- **porRightY**: Y coordinate of monocular point of regard of the right eye, also check porRightX != InvalidValue before using porRightY.
- **eyePositionRightX**: position of the right eye in device coordinates, unit is mm: More...
- **eyePositionRightY**: position of the right eye in device coordinates, unit is mm: More...
- **eyePositionRightZ**: position of the right eye in device coordinates, unit is mm: More...
- **pupilRadiusRight**: radius of the right pupil in mm

6.7 ELApi.ReturnCalibrate Class Reference

return values of calibrate( )

Static Public Attributes

- int SUCCESS = 0

  calibration successful
- int NOT_CONNECTED = 1

  cannot calibrate: not connected to the server
- int NOT_TRACKING = 2

  cannot calibrate: no device found or tracking not started
- int INVALID_CALIBRATION_MODE = 3

  cannot start calibration: calibration mode is invalid or not supported
- int ALREADY_BUSY = 4

  cannot start calibration: calibration or validation is already in progress
- int FAILURE = 5

  calibration was not successful or aborted

6.7.1 Detailed Description

return values of calibrate( )

6.8 ELApi.ReturnConnect Class Reference

return values of connect( )

Static Public Attributes

- int SUCCESS = 0
  connection successully established
- int NOT_INITED = 1
  connection failed: library needs to be initialized first (constructor call missing)
- int VERSION_MISMATCH = 2
  connection failed: API is build on a newer version than the server.
- int TIMEOUT = 3
  connection failed: the server can not be found or is not responding

6.8.1 Detailed Description

return values of connect( )

6.8.2 Member Data Documentation

6.8.2.1 VERSION_MISMATCH

```
int VERSION_MISMATCH = 2  [static]
```

connection failed: API is build on a newer version than the server.

Update the EyeLogicServer to the newest version.

6.9 ELApi.ReturnNextData Class Reference

return values of getNextEvent( ), getNextGazeSample( ) and getNextEyeImage( )

Static Public Attributes

- int SUCCESS = 0
  new event or new GazeSample received
- int NOT_INITED = 1
  library needs to be initialized first
- int TIMEOUT = 2
  timeout reached, no new event/GazeSample received
- int CONNECTION_CLOSED = 3
  connection to server closed, no new event/GazeSample received

6.9.1 Detailed Description

return values of getNextEvent( ), getNextGazeSample( ) and getNextEyeImage( )

6.10 ELApi.ReturnSetActiveScreen Class Reference

return values of setActiveScreen( )

Static Public Attributes

- int SUCCESS = 0
  active screen was set
- int NOT_FOUND = 1
  specified screen name was not found as a name of an available monitor
- int FAILURE = 2
  active screen could not be changed

6.10.1 Detailed Description

return values of setActiveScreen( )

6.11 ELApi.ReturnStart Class Reference

return values of requestTracking( )

Static Public Attributes

- int SUCCESS = 0
  start tracking successful
- int NOT_CONNECTED = 1
  not connected to the server
- int DEVICE_MISSING = 2
  cannot start tracking: no device found
- int INVALID_FRAMERATE_MODE = 3
  cannot start tracking: framerate mode is invalid or not supported
- int ALREADY_RUNNING_DIFFERENT_FRAMERATE = 4
  tracking already ongoing, but frame rate mode is different
- int FAILURE = 5
  some general failure occurred

6.11.1 Detailed Description

return values of requestTracking( )

6.12 ELApi.ReturnStreamEyeImages Class Reference

return values of streamEyeImages( )

Static Public Attributes

- int SUCCESS = 0
  setting streaming of eye images was successful
- int NOT_CONNECTED = 1
  failed, not connected to the server
- int REMOTE_CONNECTION = 2
  cannot stream eye images when connection to the server is a remote connection
- int FAILURE = 3
  failure when trying to set eye image stream

6.12.1 Detailed Description

return values of streamEyeImages( )

6.13 ELApi.ReturnValidate Class Reference

return values of validate( )

Static Public Attributes

- int SUCCESS = 0
  start validation successful
- int NOT_CONNECTED = 1
  cannot validate: not connected to the server
- int NOT_TRACKING = 2
  cannot validate: no device found or tracking not started
- int NOT_CALIBRATED = 3
  cannot start validation: validation mode is invalid or not supported
- int ALREADY_BUSY = 4
  cannot start validation: calibration or validation is already in progress
- int FAILURE = 5
  validation failure

6.13.1 Detailed Description

return values of validate( )

6.14 ELApi.ScreenConfig Class Reference

configuration of the stimulus screen

Public Member Functions

- def __init__ (self)
  constructor

Public Attributes

- localMachine
  whether this screen is connected to the local PC
- id
  ID of the screen.
- name
  Name of the screen.
- resolutionX
  screen X resolution [px]
- resolutionY
  screen Y resolution [px]
- physicalSizeX_mm
  horizontal physical dimension of the screen [mm]
- physicalSizeY_mm
  vertical physical dimension of the screen [mm]

6.14.1 Detailed Description

configuration of the stimulus screen

6.15 ELApi.ServerInfo Class Reference

connection information for an EyeLogic server

Public Member Functions

- def __init__ (self)
  constructor

Public Attributes

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

6.15.1 Detailed Description

connection information for an EyeLogic server

6.16 ELApi.ValidationPointResult Class Reference

ValidationPointResult.

Public Member Functions

- def **__init__** (self)

Public Attributes

- validationPointPxX

  ELInvalidValue or x-coordinate of stimulus point position.
- validationPointPxY

  ELInvalidValue or y-coordinate of stimulus point position.
- meanDeviationLeftPx

  ELInvalidValue or mean deviation between left eye POR and stimulus position in [px] in the stimulus plane.
- meanDeviationLeftDeg

  ELInvalidValue or mean deviation of left eye gaze direction in [deg] in the 3-D world system.
- meanDeviationRightPx

  ELInvalidValue or mean deviation between right eye POR and stimulus position in [px] in the stimulus plane.
- meanDeviationRightDeg

  ELInvalidValue or mean deviation of right eye gaze direction in [deg] in the 3-D world system.

6.16.1 Detailed Description

ValidationPointResult.

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

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

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

Note: meanDeviation∗ data fields may be ELInvalidValue. The pairs meanDeviationLeftDeg/-Px and mean↩ DeviationRightDeg-/Px are always either both valid or both ELInvalidValue.

6.17 ELApi.ValidationResult Class Reference

ValidationResult.

Public Member Functions

- def **__init__** (self)

Public Attributes

- pointsData

  Number of validation points.

6.17.1 Detailed Description

ValidationResult.

Contains one a list of ValidationPointResults - one per validation stimulus point of the performed valdation.

6.17.2 Member Data Documentation

6.17.2.1 pointsData

`pointsData`

Number of validation points.

The following arrays will hold twice this amount in valid (x, y)-tuple data points

Index