



iView X™ SDK

v4.2

June 2015

Contents

1 iView X™ API Documentation	1
1.1 Introduction	1
Welcome to the iView X™ SDK 4.2 Guide!	1
About iView X™ SDK	1
About the Guide	2
What's New?	3
System Requirements	3
Supported Eye Tracking Devices	3
Supported Programming and Scripting Languages	4
Supported Operating Systems	4
1.2 Getting Started	4
How to Set up the SDK Environment	5
Downloading	5
Running the Installer	5
Running the Demo	5
Common Workflow	6
1.3 Developing Advanced Applications	8
Single PC and Dual PC Setup	8
Connecting with Multiple Applications	11
Setting up RED Geometry	12
Binocular and Monocular Tracking Modes	15
Smart Binocular	15
Smart Tracking	15
Binocular	16
Monocular	16

Calibration	16
Accepting Calibration Points	16
Smart Calibration	17
Recalibration	17
Validation	17
Areas of Interest (AOI)	17
Polling vs. Callbacks	18
Setting the Licence Key	19
Gaze Interaction Design	19
1.4 Tutorials and Examples	20
Tutorial: Loading iViewXAPI.dll dynamically with C/C++	20
Tutorial: Displaying the gaze cursor with C#	21
Tutorial: MATLAB® Setup	25
Tutorial: Python Setup	26
Tutorial: E-Prime Setup	28
Tutorial: NBS Presentation Setup	30
Tutorial: Getting started with Unity	37
Setup	37
Get Data or Use Functions From The SDK	38
GazeMonoBehaviour	39
1.5 Frequently Asked Questions	40
Is there a 64-bit version of iViewXAPI.dll available?	40
What can I do if iV_Connect fails?	40
2 Reference	41
2.1 Enumerations	41
Detailed Description	41
Enumeration Type Documentation	41
CalibrationPointUsageStatusEnum	41
CalibrationStatusEnum	42
ETApplication	42
ETDevice	42
FilterAction	43
FilterType	43

REDGeometryEnum	43
2.2 Data Structures	44
Detailed Description	45
Data Structure Documentation	45
struct AccuracyStruct	45
struct AOIRectangleStruct	46
struct AOIStruct	46
struct CalibrationPointQualityStruct	47
struct CalibrationPointStruct	47
struct CalibrationStruct	47
struct DateStruct	48
struct EventStruct	48
struct EventStruct32	49
struct EyeDataStruct	49
struct EyePositionStruct	49
struct GazeChannelQualityStruct	50
struct ImageStruct	50
struct REDGeometryStruct	51
struct SampleStruct	51
struct SampleStruct32	52
struct SpeedModeStruct	52
struct SystemInfoStruct	52
struct TrackingStatusStruct	53
2.3 Callback Function Types	54
Detailed Description	54
2.4 Functions	55
Detailed Description	57
Function Documentation	57
iV_AbortCalibration	57
iV_AbortCalibrationPoint	58
iV_AcceptCalibrationPoint	58
iV_Calibrate	58
iV_ChangeCalibrationPoint	59

iv_ClearAOI	59
iv_ClearRecordingBuffer	60
iv_ConfigureFilter	60
iv_Connect	61
iv_ConnectLocal	61
iv_ContinueEyetracking	62
iv_ContinueRecording	62
iv_DefineAOI	62
iv_DefineAOIPort	63
iv_DeleteREDGeometry	63
iv_DisableAOI	63
iv_DisableAOIGroup	64
iv_DisableGazeDataFilter	64
iv_DisableProcessorHighPerformanceMode	64
iv_Disconnect	65
iv_EnableAOI	65
iv_EnableAOIGroup	65
iv_EnableGazeDataFilter	66
iv_EnableProcessorHighPerformanceMode	66
iv_GetAccuracy	66
iv_GetAccuracyImage	67
iv_GetAOIOutputValue	67
iv_GetAvailableLptPorts	67
iv_GetCalibrationParameter	68
iv_GetCalibrationPoint	68
iv_GetCalibrationQuality	69
iv_GetCalibrationQualityImage	69
iv_GetCalibrationStatus	69
iv_GetCurrentCalibrationPoint	70
iv_GetCurrentREDGeometry	70
iv_GetCurrentTimestamp	71
iv_GetDeviceName	71
iv_GetEvent	72

iV_GetEvent32	72
iV_GetEyeImage	72
iV_GetFeatureKey	73
iV_GetGazeChannelQuality	73
iV_GetGeometryProfiles	73
iV_GetLicenseDueDate	74
iV_GetREDGeometry	74
iV_GetSample	75
iV_GetSample32	75
iV_GetSceneVideo	75
iV_GetSerialNumber	76
iV_GetSpeedModes	76
iV_GetSystemInfo	77
iV_GetTrackingMonitor	77
iV_GetTrackingStatus	77
iV_HideAccuracyMonitor	78
iV_HideEyeImageMonitor	78
iV_HideSceneVideoMonitor	78
iV_HideTrackingMonitor	78
iV_IsConnected	79
iV_LoadCalibration	79
iV_Log	80
iV_PauseEyetracking	80
iV_PauseRecording	80
iV_Quit	81
iV_RecalibrateOnePoint	81
iV_ReleaseAOIPort	82
iV_RemoveAOI	82
iV_ResetCalibrationPoints	82
iV_SaveCalibration	83
iV_SaveData	83
iV_SelectREDGeometry	84
iV_SendCommand	84

iV_SendImageMessage	85
iV_SetAOIHitCallback	85
iV_SetCalibrationCallback	85
iV_SetConnectionTimeout	86
iV_SetEventCallback	86
iV_SetEventDetectionParameter	87
iV_SetEyeImageCallback	87
iV_SetLicense	88
iV_SetLogger	88
iV_SetREDGeometry	88
iV_SetResolution	89
iV_SetSampleCallback	89
iV_SetSceneVideoCallback	90
iV_SetSpeedMode	90
iV_SetTrackingMonitorCallback	90
iV_SetTrackingParameter	91
iV_SetupCalibration	91
iV_SetupDebugMode	92
iV_SetupLptRecording	92
iV_ShowAccuracyMonitor	93
iV_ShowEyeImageMonitor	93
iV_ShowSceneVideoMonitor	93
iV_ShowTrackingMonitor	94
iV_Start	94
iV_StartRecording	94
iV_StopRecording	95
iV_TestTTL	95
iV_Validate	96
2.5 Functions Grouped by Topic	97
2.6 Function Return Values	99
2.7 Eye Tracking Parameter	102
2.8 Functions implemented in EyeTracker2Impl for NBS Presentation	103
2.9 Function and Device Overview	105

3 Appendix	115
3.1 LICENSE AGREEMENT FOR THE SMI SOFTWARE DEVELOPMENT KITS (“SDK”) PROVIDED FREE OF CHARGE	115
3.2 Technical Support	117
3.3 About SMI	117
 Index	 118

Chapter 1

iView X™ API Documentation

1.1 Introduction

Welcome to the iView X™ SDK 4.2 Guide!

About iView X™ SDK

The iView X™ Software Development Kit ("SDK") provides an Application Interface ("API") for communication between your software application and your SMI eye tracking device, allowing you to create applications that take advantage of the powerful features offered by SensoMotoric Instruments ("SMI") eye tracking devices. Specifically, the SDK was designed for SMI customers who wish to add eye tracking into their own applications. Using the interface provided within the SDK you can control SMI eye tracking devices and retrieve eye tracking data online.

The figure below shows hard- and software components of the eye tracking system. Your application connects via the API with the iView eye tracking server.

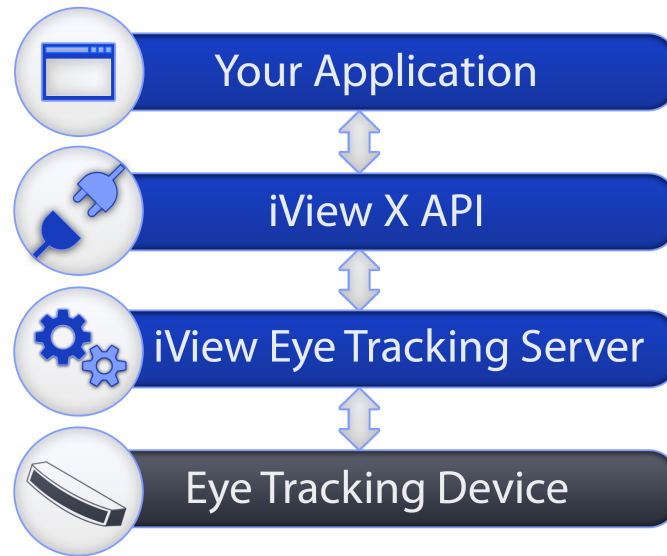


Figure 1.1: API Layers

Your Application: Your software using the API to interact with the eye tracking device. You can develop your own application or integrate 3rd party applications into your eye tracking system.

iView X™ API: Programmable interface to provide access to eye tracking device. iView X™ API is part of the iView X™ SDK. A common C Interface is provided, but you can use any programming language to build your own eye tracking application. Please check [Supported Programming and Scripting Languages](#) for details.

Eye tracking Server: iView eye tracking server application which collects the data from the eye tracking device and provides the data via the iView X™ API.

Note

Depending on your system, the iView eye tracking server functionality is provided by different binaries. For Hi-Speed, RED, etc., this is iView X. For RED-m and RED-OEM, this is the eyetracking_server. For RED250mobile, REDn Scientific and REDn Professional this is iViewNGServer. To improve readability the term **iView eye tracking server** is used as a generic name for this software component.

Eye tracking Device: eye tracking device by SMI. Please check [Supported Eye Tracking Devices](#) for a list of supported devices.

About the Guide

The SDK Guide provides a practical introduction to developing applications using the SDK and documentation about major SDK features. It includes instructions for setting up your SDK environment and a function reference, which outlines each available function. Additionally, the manual gives a brief overview on the included examples for each major platform. If you want to start working with your eye tracking device immediately, you may proceed to the section [Getting Started](#).

What's New?

If you are familiar with previous versions of the iView X™ SDK, please notice the highlights of this version:

- Tracking Modes: Access to binocular and monocular eye tracking features have been improved. The section [Binocular and Monocular Tracking Modes](#) describes the new tracking modes.
- Smart Calibration: Data quality gets assessed for each calibration point and each eye to improve gaze accuracy. Details can be found in [Smart Calibration](#).
- For some functions, new return values for a more detailed error analysis have been added.

In addition to this guide, the SDK includes release notes which can be found in the

SMI\iView X SDK\docs

directory. In the release notes you can find a complete list of the improvements and bug fixes, helping you get the most from each release.

Note

The said functionality is not available for all supported devices.

System Requirements

Note

Please note that ETG devices are not supported with this version of iView X™ SDK. Please visit <http://www.smivision.com/software> for more information.

Supported Eye Tracking Devices

The following SMI Eye Tracking Devices are supported in this release:

Supported Eye Tracking Device	Frame Rate [Hz]
HED	50 Hz / 200 Hz
HED HT	50 Hz / 200 Hz
Hi-Speed	240 Hz (monocular)
Hi-Speed	350 Hz (monocular / binocular)
Hi-Speed	500 Hz (monocular / binocular)
Hi-Speed	1250 Hz (monocular)
Hi-Speed Primate	500 Hz / 1250 Hz (monocular / binocular)
MRI LR	50 Hz

MEG	50 Hz / 250 Hz
RED 4 (Firewire)	50 Hz / 60 Hz
RED (USB)	60 Hz / 120 Hz
RED250	60 Hz / 120 Hz / 250 Hz
RED500	60 Hz / 120 Hz / 250 Hz / 500 Hz
RED-m	60 Hz / 120 Hz
RED-OEM	30 Hz - 60 Hz
RED250mobile	60 Hz / 120 Hz / 250 Hz
REDn Scientific	30 Hz / 60 Hz
REDn Professional	30 Hz / 60 Hz

Supported Programming and Scripting Languages

The iView X™ SDK can be used with most of the programming and scripting languages that are capable of importing dynamic link libraries (DLLs). These include, but are not limited to:

- C/C++
- C#
- Matlab
- E-Prime
- Python
- NBS Presentation

Supported Operating Systems

This SDK installer contains Windows 32-bit and 64-bit binaries. The SDK application files are installed into:

Type of Windows Version	Folder of iView X™ SDK
32-bit	C:\Program Files
64-bit	C:\Program Files (x86)

The iView X™ SDK for is designed to run on the following operating systems:

Operating System	Notes
Windows XP	Supported
Windows Vista 32-bit/64-bit	Supported
Windows 7 32-bit/64-bit	Supported
Windows 8, Windows 8.1 32-bit/64-bit	Supported

1.2 Getting Started

How to Set up the SDK Environment

In the following sections you will learn how to set up your SDK environment, about the various function available in the SDK, and how to create your first basic eye tracking application based on the provided examples.

Downloading

You can download the latest recommended release of the SDK from the SMI Software Downloads page: <http://www.smivision.com/software>.

Running the Installer

Note

The SDK has to be installed on the same computer as your eye tracking application. If you run your application in a Single PC setup, this will be the same computer as your iView™ software.

After you have downloaded the SDK installer package, run

SMI iView X SDK.exe

to begin the installation. When the files have been unpacked, the SDK License Agreement will appear — it contains important information about the terms under which we supply the SDK. Agree to it if you would like to proceed with the installation. If you had a previous installation it will first be removed before the new version of the SDK is installed on your computer. Please wait for the installation to complete. The installation process may take a few minutes.

Note

The SDK is already included in some RED-OEM Developer Editions, in which case there is no need to install iView X™ SDK.

Running the Demo

Once you have completed installation of the SDK, you are ready to start developing applications. To learn more about the potential of the iView X™ SDK you may start with a demo application that shows some of the features provided by the API .

To start the demo application, please

1. Connect your eye tracking device and start the eye tracking software. Depending on your device type, this is usually the iView eye tracking server.
2. Run the **HelloEyetracker.exe** application. It can be found here:

Operating System	Location
64-bit	C:\Program Files\SMI\iView X SDK\Examples\VS C#\HelloEyetracker\Application
32-bit	C:\Program Files (x86)\SMI\iView X SDK\Examples\VS C#\HelloEyetracker\Application

Press **Connect** to establish the connection between the HelloEyetracker application and the iView eye tracking server.

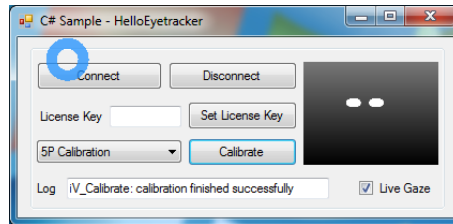


Figure 1.2: Screenshot HelloEyetracker

Once a connection has been established, tracking monitor and gaze data will be streamed automatically. You can visualize your gaze position by enabling **Live Gaze** checkbox. Gaze accuracy can be improved by performing a calibration, details about the calibration procedure can be found in the corresponding chapter [Calibration](#).

The source code for this application is available as a part of the SDK. Please have a look into the section [Tutorial: Displaying the gaze cursor with C#](#) to learn more about using C# and Microsoft Visual Studio to access the iView X™ SDK.

Common Workflow

This section describes the common workflow of eye tracking applications using the iView X™ API. In the subsequent sections you learn how to realize this workflow in your individual environment or programming language. We recommend to become familiar with the common workflow first and to study the details of your environment afterwards.

A common eye tracking application performs the following steps:

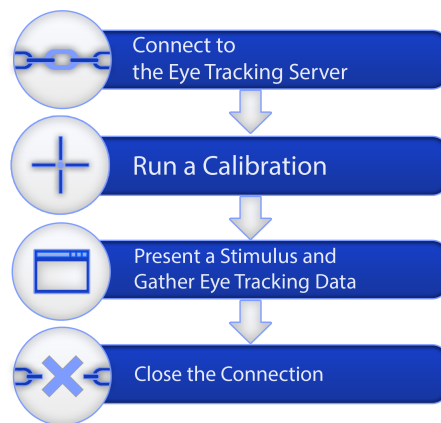


Figure 1.3: Common Workflow

For the detailed description we use a C-like programming language syntax to explain the calls to API functions. To learn how to call API functions from your preferred programming language please refer to the corresponding section.

1. Connect to the iView eye tracking server

To establish a connection call `iV_Connect`. The parameters shown here connect to iView eye tracking server running on the same PC as your application. They should work with most systems and configurations. For details of the network setup, please see [Single PC and Dual PC Setup](#) and your eye tracking device's manual.

```
iV_Connect( "127.0.0.1", 4444, "127.0.0.1", 5555);
```

After the connections have been created, the application can be used to control the iView eye tracking server's behavior or to retrieve online data for further processing.

2. Run a calibration

The next step in the common workflow is a calibration. A calibration is used to determine participant-specific physiological characteristics to initialize gaze mapping and to optimize eye tracking performance. Usually, a sequence of points is presented where the participant has to gaze at. Details about the calibration can be found in the section [Calibration](#).

```
iV_Calibrate();
```

After the calibration has been performed the system is ready to calculate and provide gaze data.

3. Present a stimulus and gather eye tracking data

There are two ways to handle eye tracking data:

Online Data Analysis: Your application retrieves and processes eye tracking data online. This can be used for interaction paradigms, e.g. gaze based control of user interfaces. The code snippet shows a loop where gaze data is polled and streamed to a console.

```
while (getchar() != 'q')  
{
```

```
SampleStruct sampleData;  
iV_GetSample( &sampleData);  
cout << "Left Eye's Gaze Data X: " << sampleData.leftEye.gazeX  
    << " Y: " << sampleData.leftEye.gazeY << endl;  
}
```

Gaze coordinates stored in **sampleData** can be used to realize gaze based interaction instead. For details about polling and other ways to retrieve online data please refer to [Polling vs. Callbacks](#).

Offline Data Analysis: Your application triggers iView eye tracking server to record eye tracking data into a file, which can be analyzed afterwards. This approach is used if data from a larger set of participants shall be analyzed or compared, or if no gaze based interaction is needed. SMI provides powerful tools for offline data analysis; please check your BeGaze manual for further information.

To start data recording, call

```
iV_StartRecording();
```

When done with recording, call

```
iV_StopRecording();
```

and finally

```
iV_SaveData( "eyedata.idf", "shortDescription", "username", 0);
```

to save the recorded data to a local file. Starting and stopping shall be synchronized with the beginning and end of your stimulus presentation.

4. Close the connection

To shutdown the connection, call

```
iV_Disconnect();
```

before closing your application.

1.3 Developing Advanced Applications

Single PC and Dual PC Setup

iView X™ API handles control flow and data flow between your application and iView eye tracking server. Control commands are submitted from your application and are addressed to the iView eye tracking server. Data is produced by the iView eye tracking server and is sent to your application. Therefore, a bidirectional connection is needed. Therefore, your application and iView eye tracking server have to configure the communication channels. Please refer to your system's manual to learn how to set up network connection at iView eye tracking server side.

For your applications, there are two ways to communicate with the iView eye tracking server via the iView X™ API:

- Single PC Setup
- Dual PC Setup

Both methods are described below.

Single PC Setup

Your application and eye tracking device are running on the same PC.

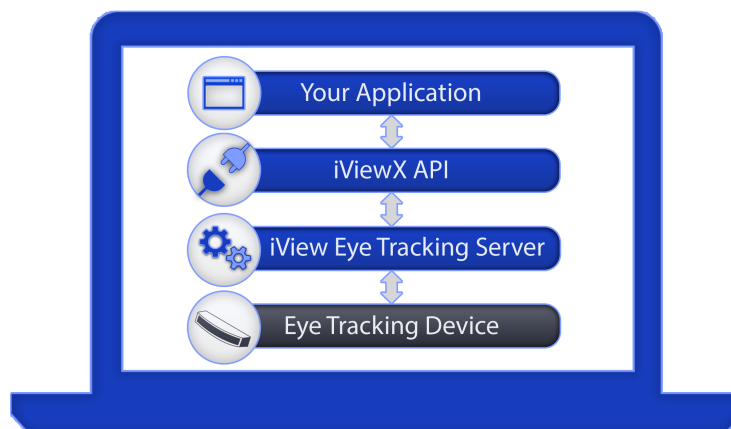


Figure 1.4: Single PC Setup

There are two ways to establish a connection between your application and the iView eye tracking server running on the same PC.

- Using [iV_ConnectLocal](#) does not require any parameters for the application or additional settings on server side.
- With [iV_Connect](#) you can establish a connection using settings for **localhost**. Please read the following instructions for details:

Although no hardware network connection is used, your application has to setup a **localhost** network connection to access iView eye tracking server. Typically, this is realized using the IP address **127.0.0.1**. The port settings have to be mirrored:

- **SendPort** from your application has to be the **ReceivePort** from iView eye tracking server. Default port number is **4444**.
- **ReceivePort** from your application has to be the **SendPort** from iView eye tracking server. Default port number is **5555**.

To establish a connection between your application and the eye tracking server, parameters of [iV_Connect](#) are:

```
iV_Connect( sendIPAddress, sendPort, recvIPAddress, receivePort);
```

In the described case **iV_Connect** has to be called from your application in the following way:

```
iV_Connect( "127.0.0.1", 4444, "127.0.0.1", 5555);
```

Note

For systems running with SMI iViewRED 4.2 or higher, it is no longer required to define the parameters **recvIPAddress** and **receivePort**. The connection can be established using

```
iV_Connect( "127.0.0.1", 4444, NULL, 0);
```

The connection has to be terminated using:

```
iV_Disconnect();
```

iV_ConnectLocal establishes a connection similar to **iV_Connect**. With **iV_ConnectLocal** port settings are handled automatically. There is no need to run **iV_Disconnect** to close a connection created with **iV_ConnectLocal**.

Dual PC Setup

Your application and iView eye tracking server are running on different PCs. Both PCs are connected via Ethernet. Low level communication between your application and iView eye tracking server via iView X™ API is realized via UDP/IP network communication.

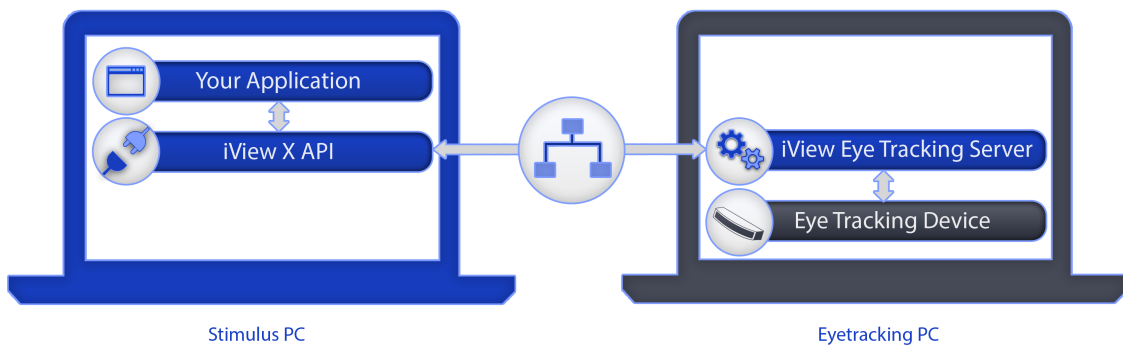


Figure 1.5: Dual PC Setup

For this example we assume the following IP addresses:

PC	IP address
Stimulus PC	192.168.1.1
Eye Tracking PC	192.168.1.2

In iView eye tracking server, the network settings have to be configured as follows:

Direction	IP address	Port
Receive/Listen	192.168.1.2	4444
Send To	192.168.1.1	5555

`iV_Connect` has to be called from your application in the following way:

```
iV_Connect ( "192.168.1.2", 4444, "192.168.1.1", 5555);
```

Note

For systems running with SMI iViewRED 4.2 or higher, it is no longer required to define the parameters **recvIPAddress** and **receivePort**. The connection can be established using

```
iV_Connect ( "192.168.1.2", 4444, NULL, 0);
```

The connection has to be terminated using:

```
iV_Disconnect ();
```

Connecting with Multiple Applications

Note

This feature is only available for RED-m and RED-OEM devices. It requires iView X™ SDK version 3.4.6 or newer and iView eye tracking server version 2.11.65 or newer.

To run multiple applications or multiple instances of the same application in parallel, each running instance has to establish its own communication channel.

The mechanism described in [Single PC and Dual PC Setup](#) allows configuration of one or at the maximum two communication channels - depending on the underlying eye tracking software's capabilities.

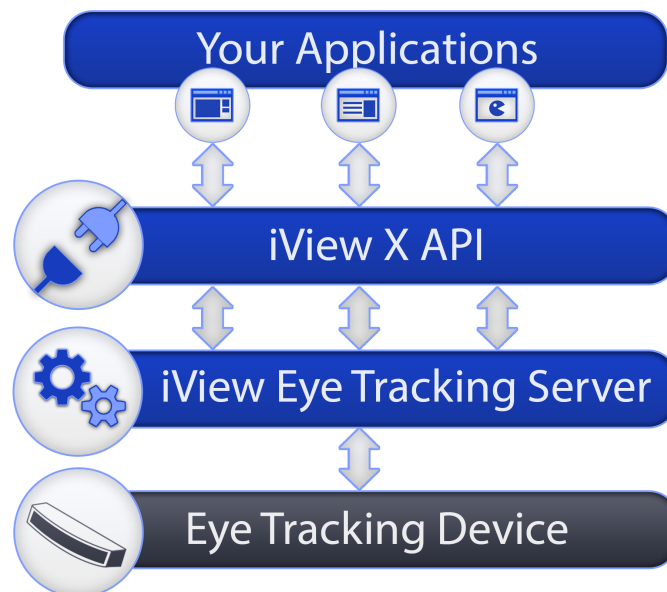


Figure 1.6: Multiple your applications on a Single PC

Setting up RED Geometry

The SDK can be used to configure the position of the RED relative to the stimulus screen. Using the proper settings for the geometry is required to reach the optimal gaze accuracy.

There are two ways to position the RED relative to the screen

- **Monitor Attached** mode describes the usage of the mounting brackets to attach the device close to the screen. This mode is available for certain devices only.
- In **Standalone** mode the RED position is independent from the stimulus screen.

Device	Available Modes
RED60, RED120, RED250, RED500	Monitor Attached, Standalone
RED-m, RED-OEM, RED250mobile, REDn Professional, REDn Scientific	Standalone

Monitor Attached Mode - RED60, RED120, RED250 and RED500

For monitor attached mode, the following parameters from the structure [REDGeometryStruct](#) are relevant:

Parameter	Value
REDGeometryStruct::redGeometry	REDGeometryEnum::monitorIntegrated
REDGeometryStruct::monitorSize	19 or 22

The function [iV_SetREDGeometry](#) configures the settings related to the display device. The monitor attached mode is not available for RED-m.

Stand Alone Mode - RED60, RED120, RED250 and RED500

The data structure [REDGeometryStruct](#) contains all required geometrical parameters. The function [iV_SetREDGeometry](#) configures the stand alone geometry.

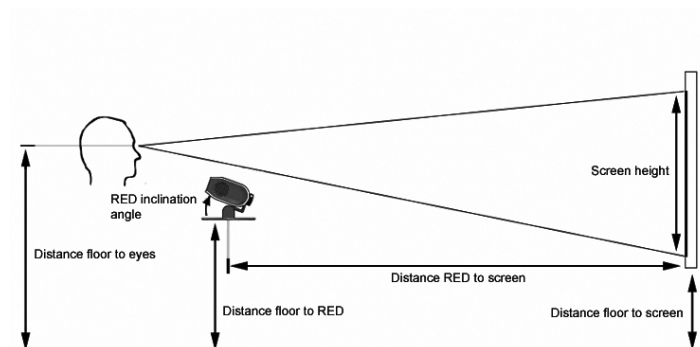


Figure 1.7: RED Stand Alone Mode

The following steps are necessary to setup the RED in stand-alone mode:

1. Remove the RED from the monitor and mount it on the stand-alone foot.
2. Position your external screen (beamer, TV, monitor) as follows:
 - The screen has to be planar
 - The screen has to be at right angle with the floor
 - The screen bottom line has to be parallel to the floor
 - RED is in the horizontal middle of the display device
3. Enter a profile name and the following geometrical dimensions of your setup into [REDGeometryStruct](#)
4. Call the function [iV_SetREDGeometry](#) including the [REDGeometryStruct](#) as parameter to iView eye tracking server

Parameter	Value
REDGeometryStruct::redGeometry	REDGeometryEnum::standalone
REDGeometryStruct::setupName	Profile name
REDGeometryStruct::stimX	Screen width [mm]
REDGeometryStruct::stimY	Screen height [mm]
REDGeometryStruct::stimHeightOverFloor	Distance floor to screen [mm]
REDGeometryStruct::redHeightOverFloor	Distance floor to RED [mm]
REDGeometryStruct::redStimDist	Distance RED to screen [mm]
REDGeometryStruct::redInclAngle	RED inclination angle [degree]

Stand Alone Mode - RED-m, RED-OEM, RED250mobile, REDn Professional and REDn Scientific

Note: Although attached to a screen, the geometrical set up has to be regarded as "stand alone" due to advanced options for configuration.

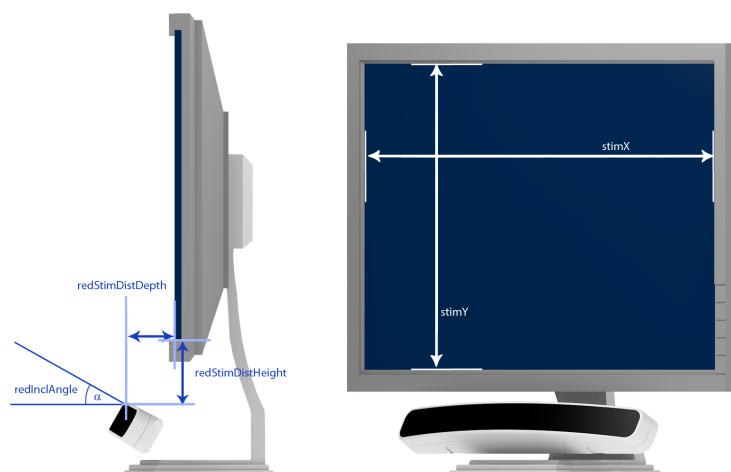


Figure 1.8: RED Stand Alone Mode

The following steps are necessary to setup the RED in stand alone mode:

1. Position your RED and your screen (beamer, TV, monitor) as follows:
 - RED is in the horizontal middle of the display device
 - Position and align the RED in a way that the user's head is in the middle of the tracking box.
2. Enter a profile name and the following geometrical dimensions of your setup into [REDGeometryStruct](#)
3. Call the function [iV_SetREDGeometry](#) including the [REDGeometryStruct](#) as parameter to iView eye tracking server

Parameter	Value
REDGeometryStruct::redGeometry	REDGeometryEnum::standalone
REDGeometryStruct::setupName	Profile name
REDGeometryStruct::stimX	Screen width [mm]
REDGeometryStruct::stimY	Screen height [mm]
REDGeometryStruct::redStimDistHeight	Vertical distance RED to stimulus screen [mm]
REDGeometryStruct::redStimDistDepth	Horizontal distance RED to stimulus screen [mm]
REDGeometryStruct::redInclAngle	RED inclination angle [degree]

Binocular and Monocular Tracking Modes

The iView X™ SDK is able to handle and setup different tracking modes which are supported by SMI RED devices. Depending on your device type, some of the modes described here may be unavailable.

Smart Binocular

The default tracking mode is **Smart Binocular** and is aimed to track and calculate the gaze of both eyes of the participant, but will tolerate if just one eye is visible to the eye tracker. In this case the system is still able to track the participant, to calculate the gaze cursor, and compensate the head movements. To enable it during run time, the following function needs to be called:

```
iV_SetTrackingParameter( ET_PARAM_EYE_BOTH,
    ET_PARAM_SMARTBINOCULAR, 0 );
```

Note

Since **SMI iViewRED 4.2**, the assignment of the left or right eye channel within the smart binocular is deprecated. `ET_PARAM_EYE_LEFT` and `ET_PARAM_EYE_RIGHT` can no longer be used with `ET_PARAM_SMARTBINOCULAR`. A similar behaviour can be accessed with using the **monocular** mode, see [Monocular](#) for details.

Smart Tracking

Smart Tracking mode is designed to optimally track participants regardless of whether both eyes are equally strong or one eye is stronger than the other. The better eye is selected if it is much better than

the other. If both eyes perform equally, binocular data is available. The function `iV_GetAccuracyImage` can be used to visualize the performance of both eyes. `iV_GetGazeChannelQuality` can be used to retrieve numerical values that allow assessing the data quality. To set up this tracking mode, call:

```
iV_SetTrackingParameter( ET_PARAM_EYE_BOTH,  
    ET_PARAM_SMARTTRACKING, 0 );
```

The eye tracker requires a validation or a calibration with at least five calibration points to assess the accuracy of the left and the right eye data channels.

Binocular

The **Binocular** tracking mode requires both eyes to be tracked. It will not tolerate one visible eye only. To enable it during run time, call:

```
iV_SetTrackingParameter( ET_PARAM_EYE_BOTH,  
    ET_PARAM_BINOCULAR, 0 );
```

Monocular

The **Monocular** mode is designed to track participants with just one active eye. Gaze is calculated for this eye only, gaze data of the other eye is ignored. Data for the active eye is written to the idf file or sent to client via the API, data channel for the other eye contains zeroed data. To set up this tracking mode, call:

To set up this tracking mode, call:

```
iV_SetTrackingParameter( ET_PARAM_EYE_RIGHT,  
    ET_PARAM_MONOCULAR, 0 );
```

for the right eye or

```
iV_SetTrackingParameter( ET_PARAM_EYE_LEFT,  
    ET_PARAM_MONOCULAR, 0 );
```

for the left eye.

Calibration

A calibration is used to determine participant-specific physiological characteristics to initialize gaze mapping and to optimize eye tracking performance. Usually, a sequence of points is presented on which the participant has to gaze at.

Accepting Calibration Points

The recommended and easiest way is to set the acceptance option to **automatically**. This mode assumes that the participant is gazing at the calibration points while they are presented. For that, the willingness of cooperation by the participant is required.

Instead of letting the server accept calibration points automatically, the participant can **manually** tell the server, by when he is fixating a calibration point. In this mode, the iView eye tracking server waits for an unlimited time for the acceptance signal from user. After receiving an acceptance signal, the server will try to track the participant gazing at the calibration point. If the participant cannot be tracked and Smart Calibration is activated, the iView eye tracking server will drop the current calibration point. Otherwise it will expect the user to accept the calibration point again, until the requirements for fixations are fulfilled.

In **Semi automatically** mode the iView eye tracking server waits for manual acceptance for the first calibration point only, the subsequent calibration points are accepted automatically.

If - for any reason - one calibration point cannot be accepted, data acquisition for that certain point can be aborted with [iV_AbortCalibrationPoint](#). The calibration procedure will continue with the subsequent point.

Smart Calibration

With **Smart Calibration** enabled, the calibration process waits for required fixations for two seconds. If any fixation is found unreliable (e.g. when the user was not really fixating that point), the fixation data will be dropped and the calibration point will not be used to calculate gaze correction parameters. This helps by removing bad fixations containing big error.

Note

When using Smart Calibration in combination with automatic acceptance of calibration points, there is a timeout of two seconds for each point. If the system is unable to track the eyes in that time, data from the current calibration point is discarded.

After a calibration in Smart Calibration mode, the client application can retrieve information about the actual usage of calibration points for each eye using [iV_GetAccuracy](#), [iV_GetAccuracyImage](#) or [iV_ShowAccuracyMonitor](#).

Recalibration

[iV_RecalibrateOnePoint](#) can be used to recalibrate a certain point in case of that participant did not fixate that point properly.

Validation

To evaluate the calibration quality the participant may perform a validation after the calibration. For that, [iV_Validate](#) has to be called. A sequence of four points is presented to the user, similar to the calibration procedure. The validation calculates the difference between the presented validation points and the measured gaze points. Overall results of the validation can be retrieved with [iV_GetAccuracy](#), [iV_GetAccuracyImage](#) or [iV_ShowAccuracyMonitor](#).

Similar to the calibration procedure [iV_AbortCalibrationPoint](#) can be used to abort the data acquisition for a validation points.

Areas of Interest (AOI)

The Area of Interest (AOI) feature allows you to define rectangular objects within the stimulus for high level gaze and fixation analysis. Your application is informed whenever the gaze enters or leaves an AOI, or a fixation event was detected within an AOI.

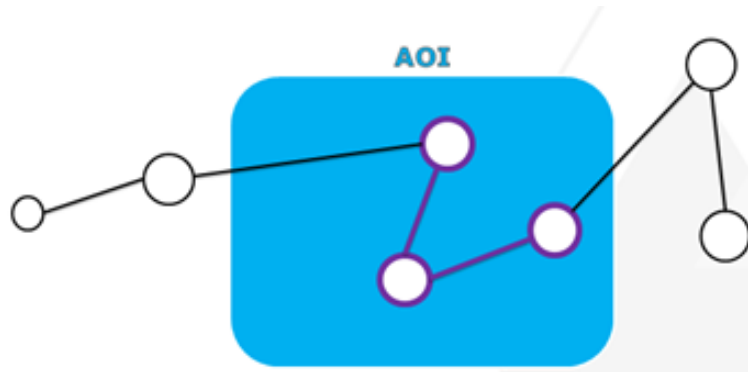


Figure 1.9: Areas of Interest

There are multiple ways to retrieve information about gaze hits or fixation hits on AOIs:

- **IDF File** If idf recording is running a message will be send to the idf data stream. This is useful if you wish to trigger and synchronize other measurement devices with the gaze position.
- **LPT Port** AOI interaction can be signaled to the LPT port. To define the port in use, call the function `iV_DefineAOIPort`. **outputValue** from `AOIStruct` can be used to define the TTL value that is send if the corresponding AOI is hit by gaze or fixation.
- **Callbacks** A Callback function can be defined that is called whenever a gaze event happens in an AOI.

For more details, see reference information for `iV_DefineAOI` and `AOIStruct` how to define AOIs.

Polling vs. Callbacks

iView X™ API provides two ways to access eye tracking data online:

- Polling
- Callbacks

The following table shows the interface functions to be used when realizing certain tasks with polling or callbacks.

Task	Polling	Callbacks
------	---------	-----------

Get event data	iV_GetEvent	iV_SetEventCallback
Get sample data	iV_GetSample	iV_SetSampleCallback
Get current calibration point	iV_GetCurrentCalibrationPoint	iV_SetCalibrationCallback
Get eye images	iV_GetEyeImage	iV_SetEyeImageCallback
Get HED scene images	iV_GetSceneVideo	iV_SetSceneVideoCallback
Get RED Tracking Monitor Image	iV_GetTrackingMonitor	iV_SetTrackingMonitorCallback
Get AOI Hits	iV_GetAOIOutputValue	iV_SetAOIHitCallback

Both methods provide different features, advantages and disadvantages. With **polling** your application has full control about the calling frequency of the polling function. Returned data will always contain the latest known values, independently if they have

- not been updated
- updated once
- updated several times

since the last call.

Callback Functions are called by the API as often as the data is updated by the underlying iView eye tracking server. Restrictions may apply due to system load.

Note

- Callback functions are not called as long as the previously executed callback of the same type has not finished. Therefore, it is recommended to put only very short and fast executing commands into callbacks.
- Callbacks are not available in all programming languages.
- Callback functions are called from different threads. Therefore, the code within callback functions has to be thread safe.
- While Polling for images (Eye Image, Tracking Monitor, Scene Video, Accuracy Image) its recommended to use only one [ImageStruct](#) instance for each data set.

Setting the Licence Key

Some SMI OEM products require a licence key to initialize communication between your application and the iView eye tracking server. Before your application opens a connection to the iView eye tracking server using [iV_Connect](#) or [iV_ConnectLocal](#), the licence key has to be passed to the system using [iV_SetLicense](#).

Gaze Interaction Design

In this chapter we want to give you some hints for gaze interaction applications and help you to create a good user experience for your users. It always helps a lot to visualize or record gaze data from your users to see, how they look in the applications or in your use case.

Keep the Natural Movement of the Eyes in Mind

Your eyes continuously scan the environment and send this input to your brain, which creates the image that you can see. Keep this basic Eye Movement rule in mind, when you design the interaction for your application. Avoid too long fixation times to activate something. Don't try to replace classic input devices like the Mouse or the Keyboard with gaze input. It feels very unnatural for your user to control your system only with your eyes. A good Gaze Interaction combines classic devices with gaze input in a smart way.

Prepare your UI Elements

To avoid wrong selections with your gaze it is important to adjust the User Interface to the needs of gaze based interactions. Create bigger AOI's around your UI Elements and use strong signal color to highlight the selection. Create enough space between the selectable Items and avoid overlapping AOIs. Be careful with animations and effects. Use them to attract the attention of your user. Avoid a detailed gaze visualization while the user interacts with your application. It may confuse the user.

Avoid the Midas-Touch Problem

It is hard for a system to distinguish between a users intention of triggering functionalities and his aim of simply exploring the user interface (UI) with his gaze. Due to this fact using eye tracking technology implies a major design compromise that is generally known as the Midas Touch Problem. Midas touch occurs when the visual exploration of the screen unintentionally activates gaze-based functionality. In order to achieve a pleasant interaction, make sure, that you trigger a gaze-based event only when the gaze remains in the area of interest (AOI) for a predefined dwell time or when a button is pressed on the controller while the user is focusing on the AOI.

1.4 Tutorials and Examples

The SDK includes sample code and applications for any major environment. All example programs described in this SDK Guide are also provided as source code in the examples directory. If you want to develop your own eye tracking application we recommend copying the example code into your development environment and use it as a starting point for your own development.

Tutorial: Loading iViewXAPI.dll dynamically with C/C++

There are two ways to integrate iViewXAPI.dll into your own C/C++ Application:

- use the Linker Library **iViewXAPI.lib** within Microsoft Visual Studio Projects.
- use late or dynamic binding to access API functions.

This tutorial describes the steps to realize the second option. You learn how to load the library iViewX-API.dll dynamically and how to access and run the function [iV_ConnectLocal](#).

First you have to declare function pointer types for the functions from iView X™ API:

```
// declare function prototype
typedef int (CALLBACK* iV_ConnectLocalType)();
```

This has to be repeated in an equivalent way for all the other functions that are used in your application. The next step is to load the iViewXAPI.dll.

```
//Load the dll and keep the handle
HINSTANCE dllHandle = NULL;
dllHandle = LoadLibrary(L"iViewXAPI.dll");
```

If this iViewXAPI.dll is loaded successfully, you have to search for the function with the corresponding name.

```
//Get pointer to our function using GetProcAddress:
iV_ConnectLocalType iV_ConnectLocalPtr = NULL;
iV_ConnectLocalPtr = (iV_ConnectLocalType)GetProcAddress( dllHandle, "
    iV_ConnectLocal");
```

Finally you can run the function:

```
// execute the function
int retVal = iV_ConnectLocalPtr();
cout << "iV_ConnectLocalPtr: " << retVal << endl;
```

Tutorial: Displaying the gaze cursor with C#

The SDK includes the source code for the C# example program **HelloEyetracker** described in [Running the Demo](#). The C# example was created using Microsoft Visual Studio 2013. This tutorial describes how to access the iView X™ API to display an overlay at the gaze position using C# and windows forms.

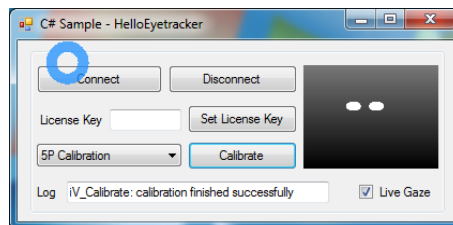


Figure 1.10: HelloEyetracker Screenshot

The first step is to integrate the iView X™ API into our own application. The following code shows how to declare external functions and data structs:

```
[DllImport("iView XAPI.dll")]
public static extern Int32 iV_Connect(StringBuilder sendIP, int
    sendPort, StringBuilder receiveIP, int receivePort);
[DllImport("iView XAPI.dll")]
public static extern Int32 iV_Disconnect();
[DllImport("iView XAPI.dll")]
public static extern Int32 iV_GetSample(ref SampleStruct
    sampleData);

public struct EyeDataStruct
{
    public double gazeX, gazeY;    // pupil gaze [pixel]
    public double diam;           // pupil diameter [pixel/mm] (mm for
        RED devices)
```

```

    public double eyePositionX      // horizontal eye position
        relative to camera (only for RED)
    public double eyePositionY      // vertical eye position
        relative to camera (only for RED)
    public double eyePositionZ;     // distance to camera (only for
        RED)
};

public struct SampleStruct
{
    public Int64 timestamp;         // timestamp of current gaze data
        sample [microseconds]
    public EyeDataStruct leftEye;   // eye data for left
        eye
    public EyeDataStruct rightEye; // eye data for left
        eye
    public Int32 planeNumber;       // plane number of gaze data
        sample (only HED HT)
};

```

The class **EyeTrackingController** which is part of the example provides an C# interface for all functions and data structures provided by iView X™ API. We recommend to use this class in your own application.

An instance of this class is generated with

```
ETDevice = new SmiSample.EyeTrackingController();
```

The following code snippets show how to use several functions from the SDK. To establish a connection with the iView eye tracking server, you first have to start the server, if it has not already been done. After the server is running, a connection to the server can be established using the **Connect** button. This button triggers the following function:

```

private void connect_Click(object sender, EventArgs e)
{
    int ret = 0;
    try
    {
        // connect to localhost server
        ret = ETDevice.iV_Connect(new StringBuilder("127.0.0.1"),
            Convert.ToInt32("4444"), new StringBuilder("127.0.0.1"), Convert.ToInt32("5555"));
        if (ret == 1) logger.Text = "iV_Connect: connection established";
        if (ret != 1) logger.Text = "iV_Connect: failed to establish
            connection: " + ret;
    }
    catch (Exception exc)
    {
        logger.Text = "Exception during iV_Connect: " + exc.Message;
    }
}

```

For some RED-OEM devices a licence key is required before a connection can be established. If you need to set a license key, enter the key in the desired text field and press the **Set License Key** button. This action will run the following function:

```

private void key_Click(object sender, EventArgs e)
{
    int ret = 0;
    try
    {
        // setting license
    }
}

```

```

    ETDevice.iV_SetLicense(new StringBuilder(licensekey.Text));
    if (ret == 1) logger.Text = "iV_SetLicense: license set successfully";
    if (ret != 1) logger.Text = "iV_SetLicense: failed to set license: " +
ret;
}
catch (Exception exc)
{
    logger.Text = "Exception during iV_SetLicense: " + exc.Message;
}
}

```

To achieve the best accuracy, each participant needs to calibrate individually. This calibration procedure can be individually changed using the combo boxes for the calibration method. To start the calibration process, click the **Calibrate** button. A calibration point will appear at the calibration area. The calibration point, which needs to be fixated by the participant, is moving from calibration point position to calibration point position until all points have been calibrated (by default 5 points).

```

private void calibrate_Click(object sender, EventArgs e)
{
    int ret = 0;
    int calibrationPoints = 5;
    int targetSize = 20;
    try
    {
        switch (calibrationMethodComboBox.Text)
        {
            case "2P Calibration":
                calibrationPoints = 2;
                break;
            default:
                case "5P Calibration":
                    calibrationPoints = 5;
                    break;
        }

        m_CalibrationData.displayDevice = 0;    // only calibrate on the main
monitor
        m_CalibrationData.autoAccept = 1;
        m_CalibrationData.method = calibrationPoints;
        m_CalibrationData.visualization = 1;
        m_CalibrationData.speed = 0;
        m_CalibrationData.targetShape = 2;
        m_CalibrationData.backgroundColor = 230;
        m_CalibrationData.foregroundColor = 250;
        m_CalibrationData.targetSize = targetSize;
        m_CalibrationData.targetFilename = "";

        ret = ETDevice.iV_SetupCalibration(ref m_CalibrationData);
        if (ret == 1) logger.Text = "iV_SetupCalibration: calibration set up
successfully";
        if (ret != 1) logger.Text = "iV_SetupCalibration: failed to setup
calibration: " + ret;
        if (ret != 1) return;

        ret = ETDevice.iV_Calibrate();
        if (ret == 1) logger.Text = "iV_Calibrate: calibration finished
successfully";
        if (ret != 1) logger.Text = "iV_Calibrate: failed to calibrate: " + ret
;
    }
    catch (System.Exception exc)
    {
        logger.Text = "Calibration Exception: " + exc.Message;
    }
}

```

```
}
```

To activate the live gaze cursor you need to check the **Live Gaze** checkbox. This will activate an additional transparent fullscreen window form to show the live gaze cursor, which has the following properties:

```
this.Opacity = 0.75f;
this.BackColor = Color.LimeGreen;    // trick to make a transparent background
this.TransparencyKey = Color.LimeGreen;
this.TopMost = true;
this.FormBorderStyle = FormBorderStyle.None;
this.WindowState = FormWindowState.Maximized;
this.DoubleBuffered = true;
this.ShowInTaskbar = false;
```

The live gaze cursor will be painted on this window form within the **OnPaint** method, which will be called every time windows needs to repaint it.

```
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    // get the tracking data
    ETDevice.iV_GetSample(ref rawDataSample);
    // average of left and right eye pos
    float posX = (float)(rawDataSample.leftEye.gazeX + rawDataSample.rightEye.
        gazeX) * 0.5f;
    float posY = (float)(rawDataSample.leftEye.gazeY + rawDataSample.rightEye.
        gazeY) * 0.5f;
    // move the pos to the middle of the circle
    posX -= circleSize/2;
    posY -= circleSize/2;
    // clamp the data to the screen resolution
    if (posX > this.Width - circleSize)
    {
        posX = this.Width - circleSize;
    }
    else if (posX < 0)
    {
        posX = 0;
    }
    if (posY > this.Height - circleSize)
    {
        posY = this.Height - circleSize;
    }
    else if (posY < 0)
    {
        posY = 0;
    }
    // draw the live gaze circle
    e.Graphics.DrawEllipse(gazePen, posX, posY, circleSize, circleSize);
}
```

You need to tell windows to update the form with the following function in your update loop:

```
liveGazeForm.Invalidate();
```

When finishing your application you need to disconnect the connection to server using the [iV_Disconnect](#) function.

```
private void disconnecting()
```

```

{
    int ret = 0;
    try
    {
        ret = ETDevice.iV_Disconnect();
        if (ret == 1) logger.Text = "iV_Disconnect: disconnected successfully";
        if (ret != 1) logger.Text = "iV_Disconnect: failed to disconnect: " +
            ret;
    }
    catch (System.Exception exc)
    {
        logger.Text = "Exception during iV_Disconnect: " + exc.Message;
    }
}

```

Tutorial: MATLAB® Setup

This tutorial describes the steps required to access iView X™ API from Matlab.

Note

To integrate iView X™ API, it is required to use a 32-bit version of MATLAB® 2014 or newer.

The examples scripts

- **Gaze Contingent Experiment:** An example that demonstrates running a calibration session and subsequently recording eye tracking data. In this experiment gaze position data is retrieved from iView eye tracking server in real time and displayed as an overlay on the presented bitmap image. The example illustrates several example functions and commands and is a good starting point for writing your own eye tracking application.

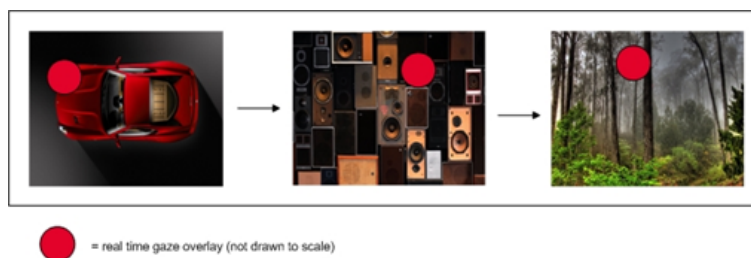


Figure 1.11: Gaze Contingent Example

- **Slide Show Experiment** runs a calibration session. Subsequently a series of images is presented to a user while eye tracking data is recorded in the background.
- **Gaze Contingent** demonstrate how to use the "psychophysics toolbox" in combination with eye tracking. Therefore it's necessary to download and install version 3.0.11 / 32-bit of the "psychophysics toolbox" from <http://psychtoolbox.org>. Read the "psychophysics toolbox" wiki for more information.

Note

The toolbox is used for visualization purposes and is not required for communication with iView eye tracking server.

For using the iView X™ SDK without the "psychophysics toolbox" have a look into the examples

- **DataStreaming**: Demonstrates how to retrieve real time eye tracking data.
- **AOIhits**: Shows how gaze hits on AOIs can be analyzed

Running the examples and using the provided source code may be a good starting point for developing own applications within MATLAB®. Within the examples there are four files that provide access to iViewXAPI.dll:

- InitViewXAPI.m
- iViewXAPI.m
- UnloadiViewXAPI.m
- InitAndConnectiViewXAPI.m

Copy those files into your own project and you are able to access iView X™ API similar to the examples.

Tutorial: Python Setup

This tutorial describes the steps required to access iView X™ API from Python.

The iView X™ SDK includes four sample experiments for use with Python. To run the experiments **Slideshow** and **GazeContingent**, it is necessary to download and install the **Psychopy toolbox**. The Psychopy toolbox is an open source toolbox that allows presentation of stimuli and collection of data for a wide range of neuroscience, psychology and psychophysics experiments. In particular, the Psychopy toolbox provides Python specific visualizations being used in these examples. Please note that the toolbox is NOT required for communication with iView eye tracking server, it is used for stimulus visualisation in the said experiments. These Python examples were written with Python version 2.7.5. and the Psychopy toolbox version 1.73.06.

Installing Prerequisites

1. Python 2.7.5 or later versions from <http://www.python.org> or any other source
2. Optional: PsychoPy Toolbox from <http://www.psychopy.org/> and additional libraries from <http://www.lfd.uci.edu/~gohlke/pythonlibs/>
or any other source
 - (a) PsychoPy Toolbox 1.73.06
 - (b) Numpy

- (c) Pyglet
- (d) Python Imaging library
- (e) wxpython
- (f) wxPython-common
- (g) Dateutil
- (h) Pyparsing

Running Examples

1. Start iView eye tracking server
2. Run Python script

Creating an Application

The following code shows how to load the required SDK DLL, connecting to iView eye tracking server, retrieving data and disconnecting from iView eye tracking server:

```
from ctypes import *

class CEye(Structure):
    _fields_ = [("gazeX", c_double),
                ("gazeY", c_double),
                ("diam", c_double),
                ("eyePositionX", c_double),
                ("eyePositionY", c_double),
                ("eyePositionZ", c_double)]

class CSample(Structure):
    _fields_ = [("timestamp", c_longlong),
                ("leftEye", CEye),
                ("rightEye", CEye),
                ("planeNumber", c_int)]

leftEye = CEye(0,0,0)
rightEye = CEye(0,0,0)
sampleData = CSample(0, leftEye, rightEye, 0)
iViewXAPI = windll.LoadLibrary("iViewXAPI.dll")
iViewXAPI.iV_Connect(c_char_p('127.0.0.1'), c_int(4444), c_char_p('127.0.0.1'), c_int(5555))
iViewXAPI.iV_GetSample(byref(sampleData))
iViewXAPI.iV_Disconnect()
```

It's recommended to use the following files as wrappers to access the iView X™ SDK.

- **iViewXAPI.py** demonstrates how to import the iView X™ SDK library and how to declare and initialize data structure that are needed for the use of the iView X™ SDK functions.
- **iViewXAPIReturnCodes.py** handles iView X™ SDK return codes.

Tutorial: E-Prime Setup

This tutorial describes the steps required to access iView X™ API from E-Prime.

The SDK includes several example experiments for E-Prime, two for the Standard version and two for the Professional version. The provided E-Prime sample experiments show you how to use this and other built-in E-Prime capabilities with the SDK functions.

The E-Prime examples were created with version 2.0.8.22 and can be converted to newer versions.

Note

The iView X™ SDK provides a package file (.epk2) for E-Prime 2 Professional to simplify the writing of your own experiments. To make the package file available in E-Prime you have to set the package's path in the E-Prime options under "Tools" → "Options..." → "Packages". In "User Search Folders:" add the following path:

```
C:\[Program Files]\SMI\iView X SDK\bin
```

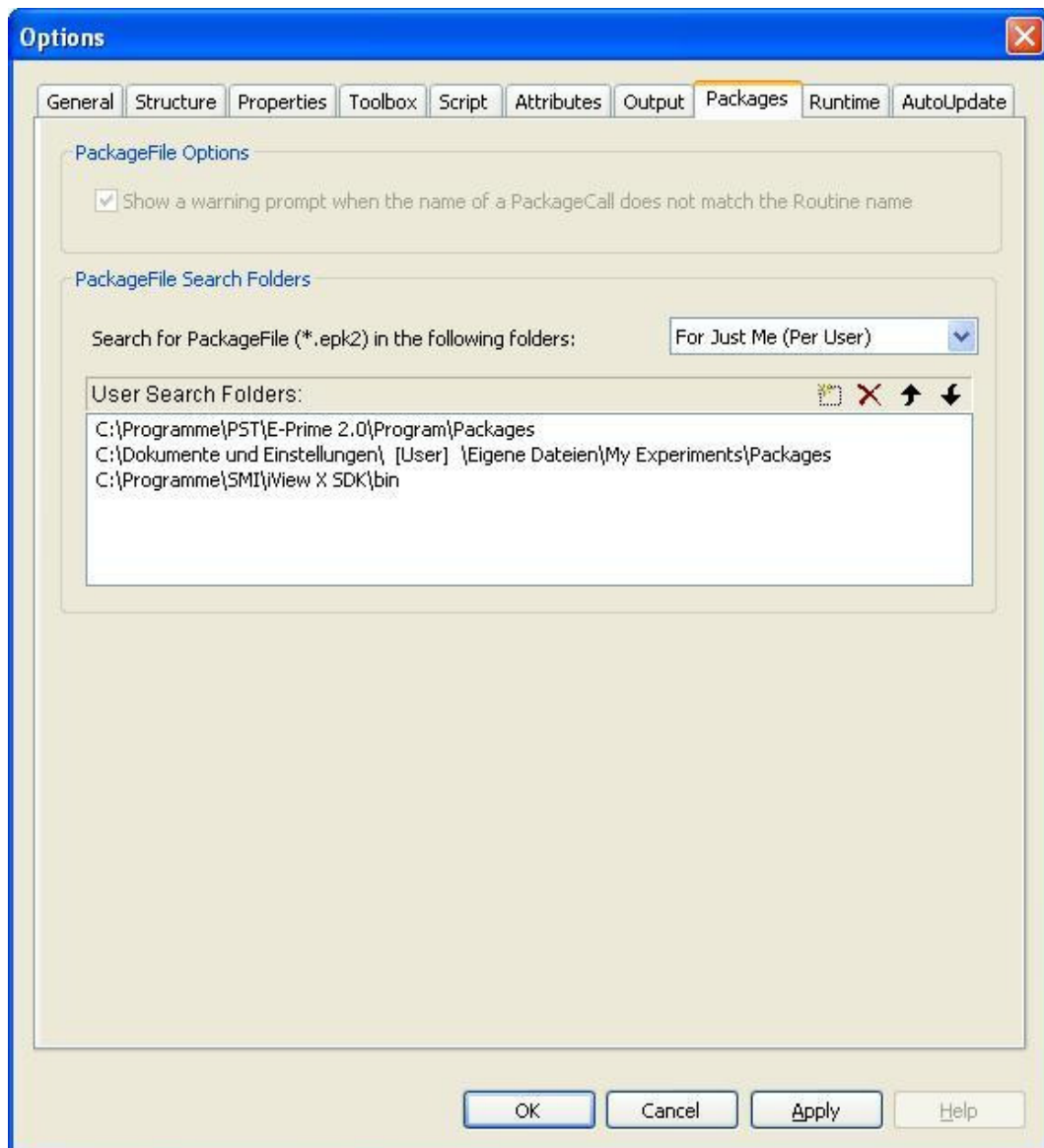


Figure 1.12: Setting up E-Prime

The following code shows how to declare structs and functions from the SDK that are needed for connecting to, getting a sample from and disconnecting from iView eye tracking server:

```

Declare Function iV_Connect Lib "iviewxapi.dll" (ByVal sendIPAddress
    As String, ByVal sendPort As Long, ByVal recvIPAddress As String, ByVal readPort
    As Long) As Long

Declare Function iV_Disconnect Lib "iviewxapi.dll" () As Long

Type EyeDataStruct
    gazeX As Double
    gazeY As Double
    diam As Double
    eyePosX As Double
    eyePosY As Double
    eyePosZ As Double
End Type

Type SampleStruct32

```

```

    timestamp As Double
    leftEye As EyeDataStruct
    rightEye As EyeDataStruct
    planeNumber As Long
End Type

Declare Function iV_GetSample32 Lib "iViewXapi.dll" (ByRef
    mySampleStruct As SampleStruct32) As Long

```

The following code shows how to connect to, get a gaze data sample and disconnect from iView eye tracking server:

```

Dim ret As Long

Dim sendIPAddress as String
Dim recvIPAddress as String
Dim sendPort As Long
Dim readPort As Long

sendPort = 4444
readPort = 5555
sendIPAddress = "127.0.0.1"
recvIPAddress = "127.0.0.1"

Dim sample As SampleStruct32

' connect to iView X
ret = iV_Connect (sendIPAddress, sendPort, recvIPAddress, readPort)

ret = iV_GetSample32 (sample)

```

It is recommended to use the custom calibration visualisation feature and to render the calibration points with E-Prime functions. To get the current calibration point's position you will need to poll for the required data using `iV_GetCurrentCalibrationPoint`. See [Polling vs. Callbacks](#) for details.

Tutorial: NBS Presentation Setup

This tutorial describes the steps required to access iView X™ API from NBS Presentation.

Since the SMI NBS Presentation extension distributes two different Presentation interfaces, both will be treated as separate objects and needs to be instantiated individually in the script file:

Interface	Class
EyeTracker2Impl	eye_tracker
PCLLibrary	iViewXAPI::eye_tracker2

NBS Presentation allows interacting with external hardware (such as eye tracking devices) using NBS Presentation extension. This extension (iViewXAPI_NBS.dll) is provided by SMI as a part of the iView X™ SDK and needs to be registered in the operation system before you can use it in the NBS Presentation experiments. There are two interfaces implemented in the delivered extension (EyeTracker2Impl and PCLLibrary) with individual functionality. While EyeTracker2Impl delivers the standard eye tracking functionality for NBS Presentation, like calibrating, validating, delivery of numerical data set, etc. the PCLLibrary extends this basic functionality by several functions which will be described below.

Registering the extension

Please follow the description below to register the NBS Presentation extension iViewXAPI_NBS.dll in Presentation:

1. In Presentation go to "Tools " → "Extension Manager".
2. In Extension Manager press "Select Extension File".
3. In the file browser that opens select the directory where iView X™ SDK is installed. Very likely this is **C:\Program Files\SMI\iView X SDK**. From this directory select subdirectory **bin**.
4. Select file **iViewXAPI_NBS.dll** and press **Open**.

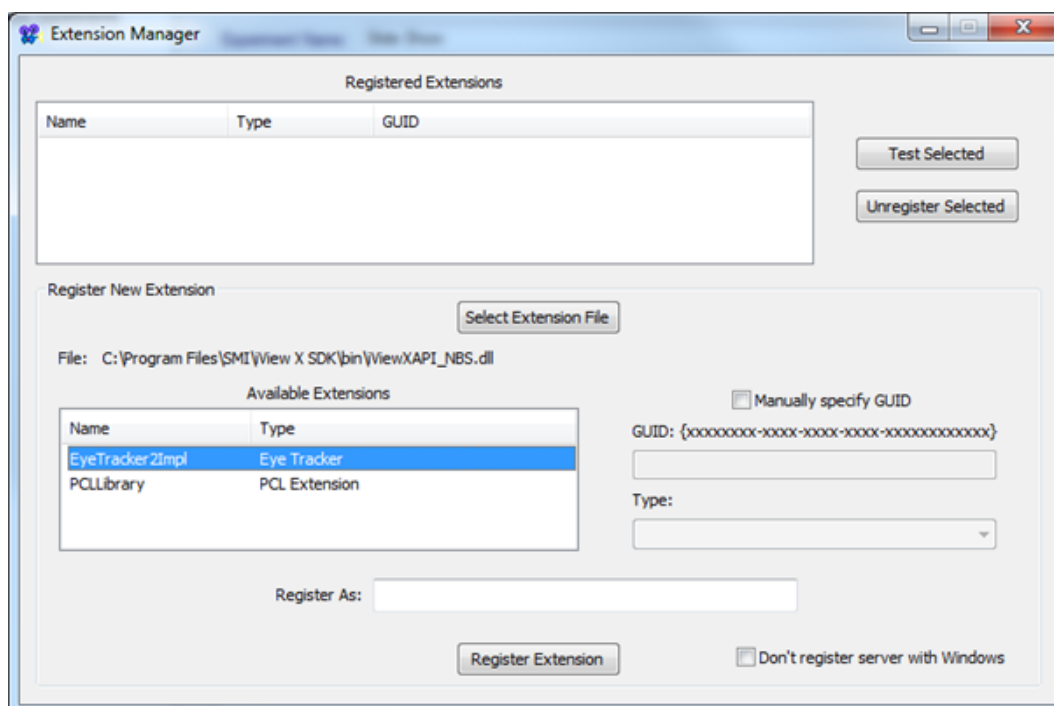


Figure 1.13: Setting up NBS Presentation, Step 4

5. In Extension Manager in **Available Extensions** select **EyeTracker2Impl**.

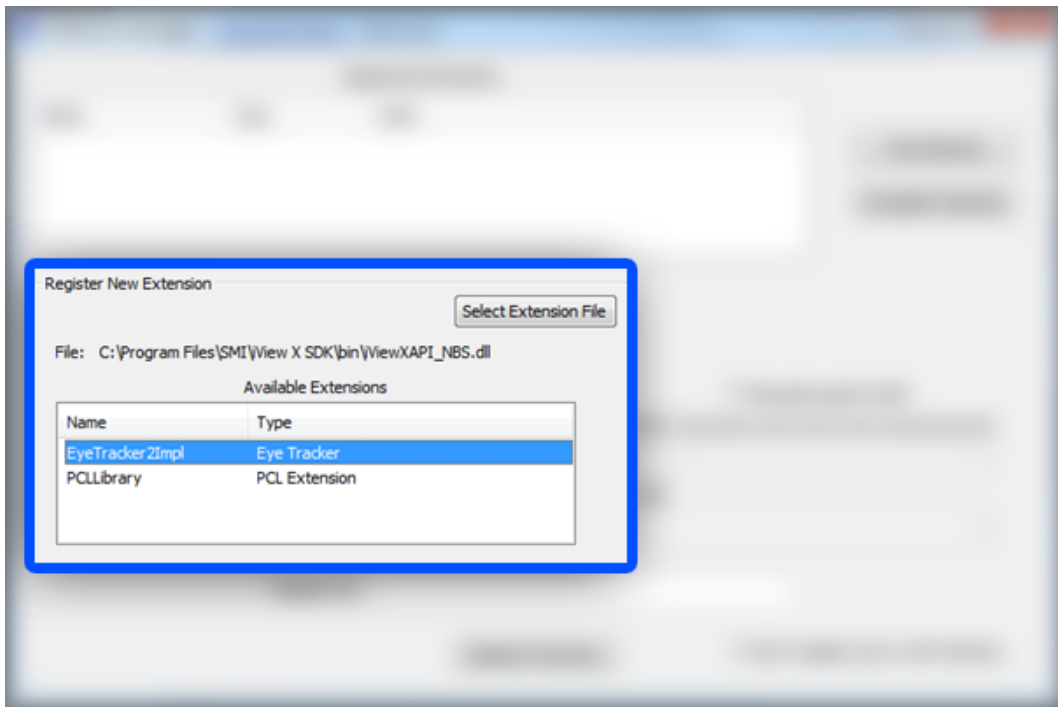


Figure 1.14: Setting up NBS Presentation, Step 5

6. In **Register As:** type "1" (or any other unique name)

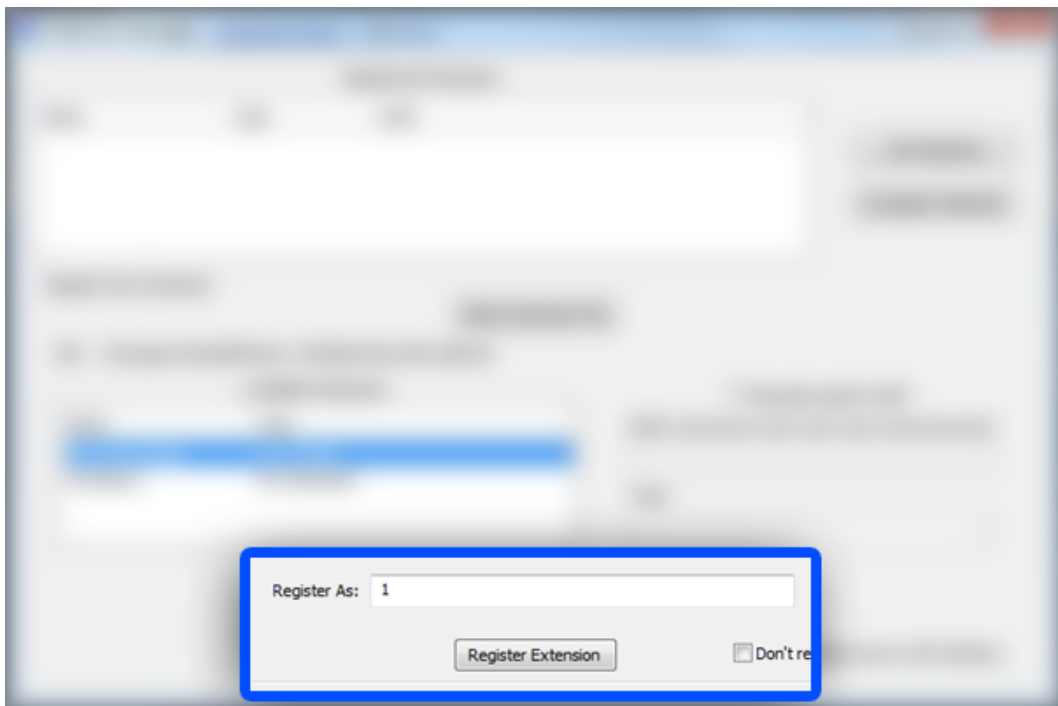


Figure 1.15: Setting up NBS Presentation, Step 6

7. Press **Register Extension**
8. Repeat steps 2 – 4.

9. In Extension Manager in **Available Extensions** select **PCLLibrary**.

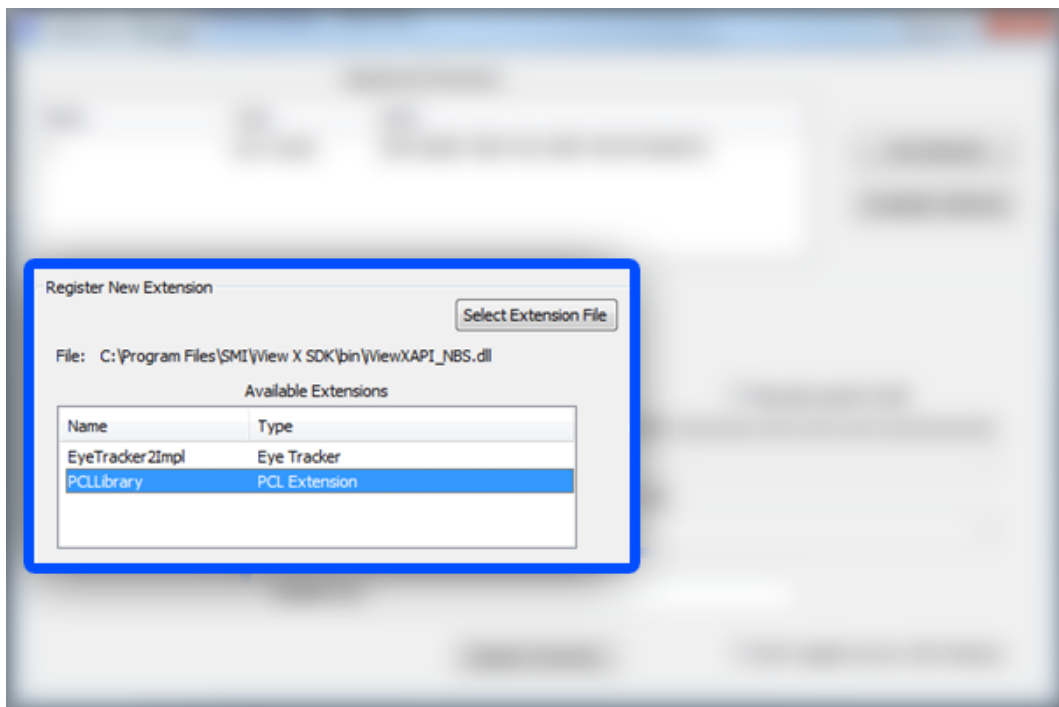


Figure 1.16: Setting up NBS Presentation, Step 9

10. In **Register As:** type "2" (or any other unique name, don't use the same name as in step 6)
11. Press **Register Extension**. Afterwards the Extension Manager should show the situation as given in the picture below:

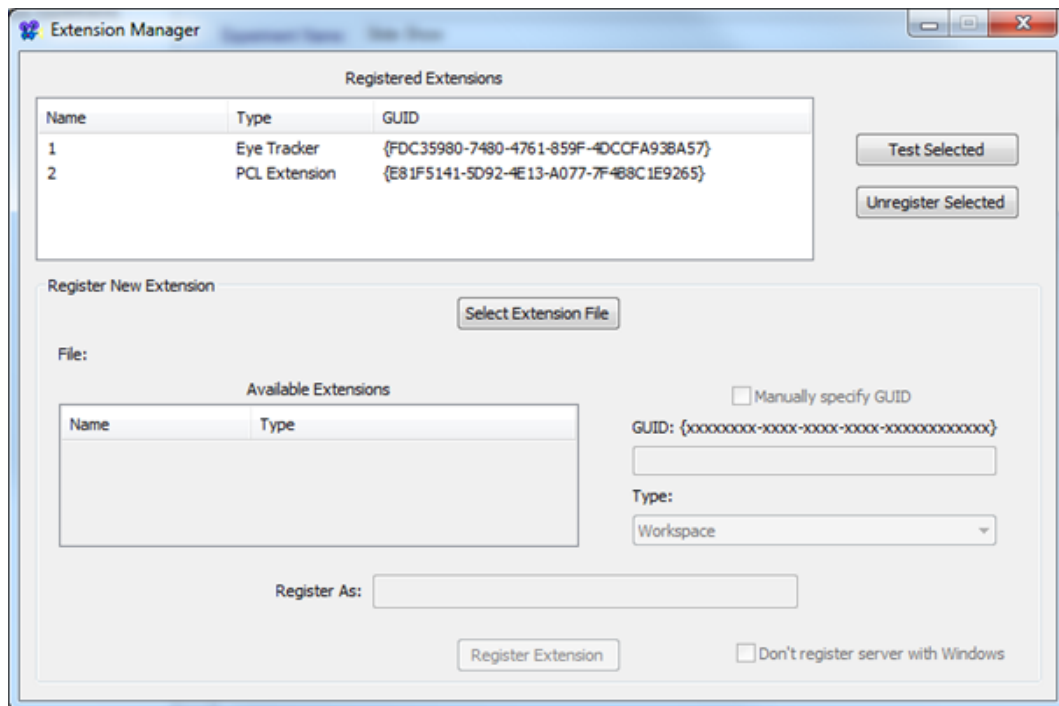


Figure 1.17: Setting up NBS Presentation, Step 11

12. Close Extension Manager. For more information on Presentation extensions and the Extension Manager please visit the NBS website <http://www.neurobs.com>.

Available Functions

EyeTracker2Impl Functionality

Functions implemented in `EyeTracker2Impl` for NBS Presentation shows which functions will be supported by the SMI EyeTracker2Impl interface. See the Presentation Help 'eye tracker extension' for function description.

PCLLibrary Functionality

```
# Establishes a connection to iView X™ (eye tracking-server).
# connect will not return until a connection has been established.
# If no connection can be established, the function will return after the
  defined time span of 3 seconds.
errorhandle connect(string sendIP, int sendport, string recvIP, int recvport)

# Disconnects from iView X™ (eye tracking-server).
# disconnect will not return until the connection has been disconnected.
# After this function has been called no other function can communicate with
  iView X™ (eye tracking-server).
errorhandle disconnect()

# Writes recorded data buffer to disc.
# The filename can include the path. If the connected eye tracking device is a
  HED, scene video buffer is written too.
# save_data will not return until the data has been saved.
errorhandle save_data()

# Returns horizontal accuracy with validated accuracy results.
```

```

# Before accuracy data is accessible the accuracy needs to be validated.
# If both eye data channels are available (binocular systems) the horizontal
  accuracy will be averaged.
# Invalid data will be marked as -1.
double get_accuracy_x()

# Returns vertical accuracy with validated accuracy results.
# Before accuracy data is accessible the accuracy needs to be validated.
# If both eye data channels are available (binocular systems) the vertical
  accuracy will be averaged.
# Invalid data will be marked as -1.
double get_accuracy_y()

```

Data handling

Due to consistency, the eye parameter handed over by functions start_data, stop_data should be equal to the parameter which will be handed over to functions like new_position_data, last_position_data, etc. and match the data which will be delivered by the SMI Eye Tracking device. If the parameters do not match the functions new_position_data might not provide any data.

Using NBS Presentation

The following code shows how to create instances of both extensions and how to use them.

```

# create PCL extension instance and connect to iView X

iViewXAPI::eye_tracker2 tracker2 = new iViewXAPI::eye_tracker2( "
    {B7A4A7F7-7879-4C95-A3BA-6CCB355AECF6}" );
tracker2.connect(iViewX_IP, Send_Port, Local_IP, Recv_Port);

# create eye tracker extension instance, start tracking and start deliver gaze
  position data

eye_tracker tracker = new eye_tracker( "{FDC35980-7480-4761-859F-4DCCFA93BA57}"
    );
tracker.start_tracking();
tracker.start_data(dt_position);

# start calibration using a predefined calibration method, acceptance and
  speed setting, and start idf data recording

tracker.calibrate( et_calibrate_default, calibration_method,
    calibration_auto_accept, calibration_speed);
tracker.set_recording (true);

# get the current gaze position data

if( tracker.new_position_data() != 0 ) then
    eyepos = tracker.last_position_data();
end;

# stop idf data recording and save the recorded data to a predefined file

tracker.set_recording(false);
tracker2.save_data("presentation_data.idf", "description", "user", 1);

# disconnect from iView

tracker2.disconnect()

```

Before getting started with the NBS Presentation example experiments included with the SDK, please verify that the following settings match your current setup:

(1) Display Device

The Display Device settings, which may be found under the **Settings** tab and Video Option, should match the actual display output setting of your environment. For example, if you will be displaying your NBS Presentation experiment on your primary monitor, the Primary Display Driver and according display mode must be selected. In the example below the display mode is 1680x1050x32 (60 Hz). If you are displaying your experiment on a secondary monitor, select the Secondary Display Driver option from the **Adapter** drop-down menu.

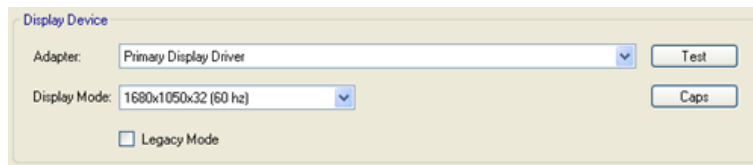


Figure 1.18: Setting up the display

(2) Screen Resolution Settings

The Screen Resolution Settings for the NBS Presentation experiments are set in the .sce file. Please make sure that the values set forth in the Display Device settings illustrated above match those in the .sce file. In the example below, the screen resolution is set to 1680x1050.

(3) Network Connection Settings

The Network Connection Settings for the NBS Presentation experiments are set in the .pcl file. Please verify that settings here match those set forth in iView X™ (Setup → Hardware → Communication → Ethernet), iView RED-m (Tray Icon → Network Connection) or iViewRED (Settings Tab). Otherwise, the NBS Presentation experiment will not be able to communicate with the Eye Tracking Server. As mentioned previously, if you are configuring your eye tracker to run in a dual PC setup, the connection settings must reflect such (i.e., the actual IP addresses and ports must be listed).

```
#####
#
#   choose connection settings to
#   establish communication with iView X
#
#####

# connection settings
string iViewX_IP = "127.0.0.1";
string Local_IP = "127.0.0.1";
int Send_Port = 4444;
int Recv_Port = 5555;
```

Note

The Presentation Interface included with the SMI iTools package does NOT need to be nor should it be used in combination with the SDK to enable communication between iViewX and NBS Presentation. In fact, they are separate packages. Communication may be enabled with NBS Presentation directly through use of the SDK. While the Presentation Interface contains useful commands for

start/stop recording and handling of the calibration process, we recommend that you use the SDK due to its more expansive feature set and capabilities.

Tutorial: Getting started with Unity

This tutorial describes the steps required to use the iView X™ API in Unity. Unity is a cross-platform game creation tool by Unity Technologies, including a 3D and 2D game engine and an integrated development environment. With Unity you can create various applications like games, simulations or training-software. The Unity extension of the iView X™ SDK works with Unity Free and Unity Pro.

You can download the newest Version of Unity here: <http://unity3d.com/unity/download>

The System Requirements for Unity can be found here: <http://unity3d.com/unity/system-requirements>

Note

The Unity extension of the iView X™ SDK is designed for the Unity Version 4.6.

Setup

The package **SML_iviewXintegration.package** contains only the Unity integration of the iView X™ SDK. The package **SML_iviewXintegration_Example.package** contains the Unity integration and examples.

Let's start with the examples: Import the first package into an empty Project. Now you will see the following folders in your Project:

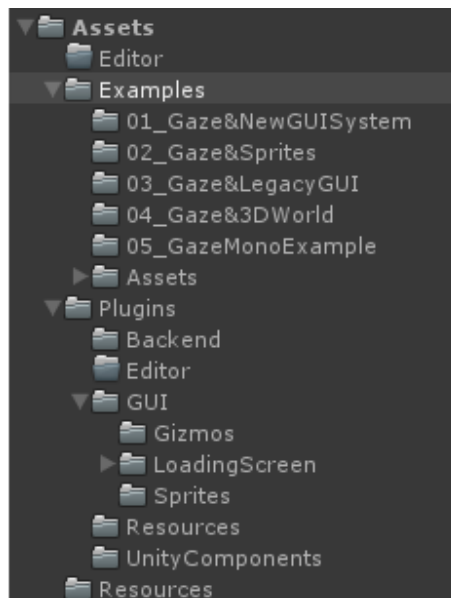


Figure 1.19: Unity Project Folder

To run an example just open one of the Example scenes and press play. The following table shows you the content of the Examples:

Name	Content
01_Gaze&NewGUISystem	Shows how to use the gaze with the new GUI-System (since Unity 4.6). The icons will scale, when the gaze hits them.
02_Gaze&Sprites	Similar to the first example, but internally using the Unity sprite engine for the interaction.
03_Gaze&LegacyGUI	Uses the legacy GUI-System of Unity (before Unity 4.6). The icon will change, when the gaze hits it.
04_Gaze&3DWorld	Shows how to use the gaze in the Unity 3D world. The cubes will scale, when the gaze hits the object.
05_Gaze&MonoExample	Example for the class GazeMonoBehaviour. Look onto one of the boxes and hit the spacebar to shoot the boxes.

All examples provide the following keyboard shortcuts:

- Press 2 to start a two Point Calibration.
- Press 5 to start a five Point Calibration.
- Press 3 to start a Validation.

You can change the Default Inputkeys in the Inspector and in the SMIGazecontroller.cs class:

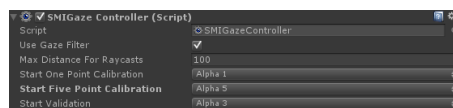


Figure 1.20: SMIGazeController properties

Get Data or Use Functions From The SDK

Let's have a look how to create your own content. The following example shows, how you access the Data from the Eye Tracker:

```
//get the Sample from the Server
SampleData sample = SMIGazeController.Instance.GetSample();

//get the averaged GazePosition
Vector3 averageGazePosition = sample.averagedEye.
    gazePositionInUnityScreenCoordinatespace();
```

The SDK provides the informations in three coordinatespaces:

Name	Description
------	-------------

<code>gazePosInUnityScreenCoords ();</code>	Returns the position of the Gaze in the Unity-Screencoordinate System. The (0,0)-Point is down left. (in px)
<code>gazePosInScreenCoords();</code>	Returns the position of the Gaze in the Windows-Screencoordinate System. The (0,0)-Point is top left. (in px)
<code>gazePosInViewPortCoords()</code>	Returns the position of the Gaze in the Windows-Screencoordinate System. The (0,0)-Point is top left. (in px)

Now you can use this vector to raycast into the scene and get e.g. the name of the selected Object:

```
Ray rayGaze = Camera.main.ScreenPointToRay(averageGazePosition);
RaycastHit hit;

//Raycast from the Gaze position on the Screen
if(Physics.Raycast(rayGaze,out hit))
{
    //Print the name of the Object in the raygaze
    Debug.Log("The name of the Object is: " + hit.collider.gameObject.name);
}
```

The Unity Integration provides a simple way to get the current selection of the gaze. The following example shows you how you can get the focused object:

```
GameObject objectInFocus = SMIGazeController.Instance.GetObjectInFocus(
    FocusFilter.WorldSpaceObjects);
```

If you want to start a calibration or a validation just use this snippet.

```
int calibrationType = 5;

// Start a calibration
SMIGazeController.Instance.StartCalibration(calibrationType);

// Start a validation
SMIGazeController.Instance.StartValidation();
```

Note

Please have a look into the iView eye tracking software's manual, which calibration types are available for your device.

GazeMonoBehaviour

We prepared a basic Monobehaviour with basic functions for gaze interaction applications. Just derive your own class from the class `GazeMonoBehaviour` and overwrite the basic functions (`OnGazeEnter`, `OnGazeStay`, `OnGazeExit`). Use the functions to define how you want to react, if the gaze hits the gameobject or has left the gameobject. You can use the `RaycastHit` in the function to get more information about the Raycast. Please have a look into the Unity Documentation for more information about the `RaycastHit` class <http://docs.unity3d.com/ScriptReference/RaycastHit.html>

The following example shows you how it works:

```
using UnityEngine;
using System.Collections;

//Make sure that you use the namespace
using iView;

//Extend the Script from the GazeMonobehaviour
public class GazeExample_GazeMonoExample : GazeMonobehaviour
{
    public override void OnGazeEnter(RaycastHit hitInformation)
    {
        //Triggers, when the Gaze hits for the first time into the object
        base.OnGazeEnter(hitInformation);
    }

    public override void OnGazeStay(RaycastHit hitInformation)
    {
        //Triggers, when the Gaze hits stays in the Object
        base.OnGazeStay(hitInformation);
    }

    public override void OnGazeExit()
    {
        //Triggers, when the Gaze leave the Object
        base.OnGazeExit();
    }
}
```

1.5 Frequently Asked Questions

- Is there a 64-bit version of iViewXAPI.dll available?
- What can I do if iV_Connect fails?

Is there a 64-bit version of iViewXAPI.dll available?

With this version of the iView X™ SDK, there is no 64-bit version of iViewXAPI.dll available.

What can I do if iV_Connect fails?

- Check, if eye tracking_server is running. You may use the iView client to check if your system works correctly.
- Check, if you entered the correct licence using [iV_SetLicense](#).

Chapter 2

Reference

2.1 Enumerations

Enumerations

- enum `CalibrationPointUsageStatusEnum` {
 `calibrationPointUsed` = 0, `calibrationPointUnused` = 1, `calibrationPointUnusedBecauseOfTimeout` = 2, `calibrationPointUnusedBecauseOfBadQuality` = 3,
 `calibrationPointIgnored` = 4 }
- enum `CalibrationStatusEnum` { `calibrationUnknown` = 0, `calibrationInvalid` = 1, `calibrationValid` = 2,
 `calibrationInProgress` = 3 }
- enum `ETApplication` { `iViewX` = 0, `iViewXOEM` = 1, `iViewNG` = 2 }
- enum `ETDevice` {
 `NONE` = 0, `RED` = 1, `REDm` = 2, `HiSpeed` = 3,
 `MRI` = 4, `HED` = 5, `Custom` = 7, `REDn` = 8 }
- enum `FilterAction` { `Query` = 0, `Set` = 1 }
- enum `FilterType` { `Average` = 0 }
- enum `REDGeometryEnum` { `monitorIntegrated` = 0, `standalone` = 1 }

Detailed Description

Enumeration Type Documentation

enum `CalibrationPointUsageStatusEnum`

This enum provides information about whether a calibration point was actually used for the calibration or the reason why it was not used. Use `iV_GetCalibrationQuality` to retrieve the calibration quality data.

Enumerator:

`calibrationPointUsed` the calibration point was accepted and is used for the calibration

`calibrationPointUnused` the calibration point was rejected and is not used for the calibration

calibrationPointUnusedBecauseOfTimeout the calibration point was rejected because no fixation was detected within reasonable time

calibrationPointUnusedBecauseOfBadQuality the calibration point was rejected because the detected fixation was too imprecise

calibrationPointIgnored no fixation was detected for the calibration point, but it is also not required

enum CalibrationStatusEnum

This enum provides information about the iView eye tracking server calibration status. If the device is not calibrated the iView eye tracking server won't deliver valid gaze data. Use the functions [iV_GetCalibrationStatus](#) to retrieve the calibration status and [iV_Calibrate](#) to perform a calibration.

Enumerator:

calibrationUnknown calibration status is unknown (i.e. if the connection is not established)

calibrationInvalid the device is not calibrated and will not deliver valid gaze data

calibrationValid the device is calibrated and will deliver valid gaze data

calibrationInProgress the device is currently performing a calibration

enum ETApplication

ETApplication can be used to start iView X, iView X OEM or iViewNGServer (iView eye tracking server) application dependent to the used eye tracking device. Set this as a parameter in [iV_Start](#) function.

Enumerator:

iViewX for iView X based devices like RED, HiSpeed, MRI, HED

iViewXOEM for RED-OEM based devices like RED-m or other customized RED-OEM devices

iViewNG for RED250mobile and REDn devices

enum ETDevice

The enumeration ETDevice can be used in connection with [iV_GetSystemInfo](#) to get information about which type of device is connected to iView X or iView eye tracking server. It is part of the [SystemInfoStruct](#).

Enumerator:

NONE if no device is set up while running iView X application

RED iView X based remote eye tracking devices

REDm eye tracking server based remote eye tracking devices

HiSpeed iView X based hi speed eye tracking devices

MRI iView X based MRI eye tracking devices

HED iView X based head mounted eye tracking devices

Custom iView X based custom devices like the mouse grabber

REDn iViewNG based RED250mobile and REDn eye tracking devices

enum FilterAction

FilterType can be used to select the action that is performed when calling [iV_ConfigureFilter](#).

Enumerator:

Query query the current filter status

Set configure filter parameters

enum FilterType

FilterType can be used to select the filter that is used with [iV_ConfigureFilter](#).

Enumerator:

Average left and right gaze data channels are averaged the type of the parameter data from [iV_ConfigureFilter](#) has to be converted to int*. The value of data can be [0;1] where 0 means averaging is disabled and 1 means averaging is enabled

enum REDGeometryEnum

uses to the define the content of [REDGeometryStruct](#)

Enumerator:

monitorIntegrated use monitor integrated mode

standalone use standalone mode

2.2 Data Structures

Data Structures

- struct [AccuracyStruct](#)

This struct provides information about the last validation. Therefore a valid validation must be successfully completed before the [AccuracyStruct](#) can be updated. To update information in [AccuracyStruct](#) use function [iV_GetAccuracy](#). [More...](#)

- struct [AOIRectangleStruct](#)

Use this struct to customize the AOI position on screen. [AOIRectangleStruct](#) is a part of [AOIStruct](#) and can be defined with [iV_DefineAOI](#). [More...](#)

- struct [AOIStruct](#)

Use this struct to customize trigger AOIs. To define AOIs on screen, trigger parameter and trigger values use [iV_DefineAOIPort](#) and [iV_DefineAOI](#) functions. [More...](#)

- struct [CalibrationPointQualityStruct](#)

This struct provides information about the fixation quality when a calibration point was shown. If the request calibration quality data is not available, the number and positionX/positionY will be set to -1. User have to check these fields to make sure the returned data is valid. [More...](#)

- struct [CalibrationPointStruct](#)

This struct provides information about the position of calibration points. To update information in [CalibrationPointStruct](#) during a calibration or validation use function [iV_GetCurrentCalibrationPoint](#). Before or after the calibration use [iV_GetCalibrationPoint](#). [More...](#)

- struct [CalibrationStruct](#)

Use this struct to customize the calibration and validation behavior. To set calibration parameters with [CalibrationStruct](#) use function [iV_SetupCalibration](#) before a calibration or validation is started. [More...](#)

- struct [DateStruct](#)

Use this struct to get the license due date of the device. Use the function [iV_GetLicenseDueDate](#) to update information in [DateStruct](#). [More...](#)

- struct [EventStruct](#)

This struct provides information about the last eye event that has been calculated. To update information in [EventStruct](#) use function [iV_GetEvent](#) or set the event callback with [iV_SetEventCallback](#). [More...](#)

- struct [EventStruct32](#)

This struct provides information about the last eye event that has been calculated. The difference to [EventStruct](#) is that the timestamp will be stored in milliseconds instead of microseconds and the order of the components are different. To update information in [EventStruct32](#) use function [iV_GetEvent32](#). [More...](#)

- struct [EyeDataStruct](#)

This struct provides numerical information about eye data. [EyeDataStruct](#) is part of [SampleStruct](#). To update information in [SampleStruct](#) use function [iV_GetSample](#) or set the sample callback with [iV_SetSampleCallback](#). [More...](#)

- struct [EyePositionStruct](#)

This value represents the relative position of the eye in the tracking box. The 0 is defined at the center position. The value +1 defines the upper/right/far maximum while the value -1 the lower/left/near maximum. The position rating is related to the tracking monitor and represents how critical the tracking and the position is, related to the border of the tracking box. The 0 is defined as the best eye position to be tracked

while the value +1 defines that the eye is almost not being tracked due to extreme upper/right/far position. The value -1 defines that the eye is almost not being tracked due to extreme lower/left/near position. If the eye isn't tracked at all the validity flag goes to 0 and all values for the represented eye will be set to 0.
[More...](#)

- struct [GazeChannelQualityStruct](#)

This struct provides information about the last validation. Therefore a valid validation must be successfully completed before the [GazeChannelQualityStruct](#) can be updated. To update information in [GazeChannelQualityStruct](#) use function [iV_GetGazeChannelQuality](#). [More...](#)

- struct [ImageStruct](#)

Use this struct to get raw eye image, raw scene video image, or raw tracking monitor image. For receiving raw eye image (format: monochrome 8bpp) use [iV_GetEyeImage](#), or set the eye image callback with [iV_SetEyeImageCallback](#). For receiving raw scene video image (format: RGB 24bpp) use [iV_GetSceneVideo](#), or set the scene video callback with [iV_SetSceneVideoCallback](#). For receiving raw tracking monitor image (format: RGB 24bpp) use [iV_GetTrackingMonitor](#), or set the tracking monitor callback with [iV_SetTrackingMonitorCallback](#). [More...](#)

- struct [REDGeometryStruct](#)

Use this struct to customize the RED geometry. See chapter [Setting up RED Geometry](#) in the iView X SDK Manual for details. For setting up the RED geometry parameters with [REDGeometryStruct](#) use function [iV_SetREDGeometry](#). [More...](#)

- struct [SampleStruct](#)

This struct provides information about an eye data sample. To update information in [SampleStruct](#) use the function [iV_GetSample](#) or set the sample callback with [iV_SetSampleCallback](#). [More...](#)

- struct [SampleStruct32](#)

This struct provides information about a eye data samples. To update information in [SampleStruct32](#) use the function [iV_GetSample32](#). The difference to [SampleStruct](#) is that the timestamp will be stored in milliseconds instead of microseconds. [More...](#)

- struct [SpeedModeStruct](#)

This struct provides information about the speed modes used and supported by the connected iView eye tracking server. To update information in [SpeedModeStruct](#) use function [iV_GetSpeedModes](#). [More...](#)

- struct [SystemInfoStruct](#)

This struct provides information about the iView eye tracking server version and the API version in use. To update data in [SystemInfoStruct](#) use the function [iV_GetSystemInfo](#). [More...](#)

- struct [TrackingStatusStruct](#)

This struct provides information about the relative eye ball position within the tracking box. The information will be provided for each eye individually as well as for the geographical center between both eyes. To update information in [TrackingStatusStruct](#) use the function [iV_GetTrackingStatus](#). [More...](#)

Detailed Description

Data Structure Documentation

struct [AccuracyStruct](#)

This struct provides information about the last validation. Therefore a valid validation must be successfully completed before the [AccuracyStruct](#) can be updated. To update information in [AccuracyStruct](#) use

function [iV_GetAccuracy](#).

Data Fields

double	deviationLX	horizontal calculated deviation for left eye [degree]
double	deviationLY	vertical calculated deviation for left eye [degree]
double	deviationRX	horizontal calculated deviation for right eye [degree]
double	deviationRY	vertical calculated deviation for right eye [degree]

struct AOIRectangleStruct

Use this struct to customize the AOI position on screen. [AOIRectangleStruct](#) is a part of [AOIStruct](#) and can be defined with [iV_DefineAOI](#).

Data Fields

int	x1	x-coordinate of left border of the AOI [pixel]
int	x2	x-coordinate of right border of the AOI [pixel]
int	y1	y-coordinate of upper border of the AOI [pixel]
int	y2	y-coordinate of lower border of the AOI [pixel]

struct AOIStruct

Use this struct to customize trigger AOIs. To define AOIs on screen, trigger parameter and trigger values use [iV_DefineAOIPort](#) and [iV_DefineAOI](#) functions.

Data Fields

char	aoiGroup	group name of AOI
char	aoiName	name of AOI
int	enabled	enable/disable trigger functionality [1: enabled, 0: disabled]
char	eye	['l', 'r']
int	fixationHit	uses fixations or raw data as trigger [1: fixation hit, 0: raw data hit]
char	output-Message	message in idf data stream
int	outputValue	TTL output value.
struct AOIRectangle-Struct	position	position of AOI

struct CalibrationPointQualityStruct

This struct provides information about the fixation quality when a calibration point was shown. If the request calibration quality data is not available, the number and positionX/positionY will be set to -1. User have to check these fields to make sure the returned data is valid.

Use [iV_GetCalibrationQuality](#) to retrieve the calibration quality data

Data Fields

double	correctedPorX	horizontal position of corrected fixation point [pixel]
double	correctedPorY	vertical position of corrected fixation point [pixel]
int	number	number of calibration point
int	positionX	horizontal position of calibration point [pixel]
int	positionY	vertical position of calibration point [pixel]
double	qualityIndex	quality index indicates how likely the user was really fixating on the calibration point when it was shown. It has a value between 0 and 1. The higher the value, the more likely the user was fixating the calibration point as required.
double	standard-DeviationX	horizontal standard deviation of the gaze samples, which represents the noise level of the fixation, given in [pixel]
double	standard-DeviationY	vertical standard deviation of the gaze samples, which represents the noise level of the fixation, given in [pixel]
enum Calibration-PointUsage-StatusEnum	usageStatus	a flag indicating whether the calibration point was really used for the calibration or the reason why it was not used.

struct CalibrationPointStruct

This struct provides information about the position of calibration points. To update information in [CalibrationPointStruct](#) during a calibration or validation use function [iV_GetCurrentCalibrationPoint](#). Before or after the calibration use [iV_GetCalibrationPoint](#).

Data Fields

int	number	number of calibration point
int	positionX	horizontal position of calibration point [pixel]
int	positionY	vertical position of calibration point [pixel]

struct CalibrationStruct

Use this struct to customize the calibration and validation behavior. To set calibration parameters with [CalibrationStruct](#) use function [iV_SetupCalibration](#) before a calibration or validation is started.

Data Fields

int	autoAccept	set calibration/validation point acceptance [2: full-automatic, 1: semi-automatic (default), 0: manual]
int	background-Brightness	set calibration/validation background brightness [0..255] (default: 220)
int	displayDevice	set display device [0: primary device (default), 1: secondary device]
int	foreground-Brightness	set calibration/validation target brightness [0..255] (default: 250)
unsigned int	method	select calibration method (default: 5) A bit mask is used to specify a new calibration workflow. If the highest bit is 1, The "Smart Calibration" workflow should be activated on iView eye tracking server.
int	speed	set calibration/validation speed [0: slow (default), 1: fast]
char	targetFilename	select custom calibration/validation target (only if targetShape = 0)
int	targetShape	set calibration/validation target shape [IMAGE = 0, CIRCLE1 = 1, CIRCLE2 = 2 (default), CROSS = 3]
int	targetSize	set calibration/validation target size (default: 20 pixels)
int	visualization	draw calibration/validation by API (default: 1)

struct DateStruct

Use this struct to get the license due date of the device. Use the function [iV_GetLicenseDueDate](#) to update information in [DateStruct](#).

Data Fields

int	day	day of license expiration
int	month	month of license expiration
int	year	year of license expiration

struct EventStruct

This struct provides information about the last eye event that has been calculated. To update information in [EventStruct](#) use function [iV_GetEvent](#) or set the event callback with [iV_SetEventCallback](#).

Data Fields

long long	duration	duration of the event [microseconds]
long long	endTime	end time of the event [microseconds]
char	eventType	type of eye event, 'F' for fixation (only fixations are supported)
char	eye	related eye, 'l' for left eye, 'r' for right eye
double	positionX	horizontal position of the fixation event [pixel]

double	positionY	vertical position of the fixation event [pixel]
long long	startTime	start time of the event [microseconds]

struct EventStruct32

This struct provides information about the last eye event that has been calculated. The difference to [EventStruct](#) is that the timestamp will be stored in milliseconds instead of microseconds and the order of the components are different. To update information in [EventStruct32](#) use function [iV_GetEvent32](#).

Data Fields

double	duration	duration of the event [milliseconds]
double	endTime	end time of the event [milliseconds]
char	eventType	type of eye event, 'F' for fixation (only fixations are supported)
char	eye	related eye, 'l' for left eye, 'r' for right eye
double	positionX	horizontal position of the fixation event [pixel]
double	positionY	vertical position of the fixation event [pixel]
double	startTime	start time of the event [milliseconds]

struct EyeDataStruct

This struct provides numerical information about eye data. [EyeDataStruct](#) is part of [SampleStruct](#). To update information in [SampleStruct](#) use function [iV_GetSample](#) or set the sample callback with [iV_SetSampleCallback](#).

Data Fields

double	diam	pupil diameter [mm]
double	eyePositionX	horizontal eye position relative to camera [mm]
double	eyePositionY	vertical eye position relative to camera [mm]
double	eyePositionZ	distance to camera [mm]
double	gazeX	horizontal gaze position on screen [pixel]
double	gazeY	vertical gaze position on screen [pixel]

struct EyePositionStruct

This value represents the relative position of the eye in the tracking box. The 0 is defined at the center position. The value +1 defines the upper/right/far maximum while the value -1 the lower/left/near maximum. The position rating is related to the tracking monitor and represents how critical the tracking and the position is, related to the border of the tracking box. The 0 is defined as the best eye position to be tracked while the value +1 defines that the eye is almost not being tracked due to extreme upper/right/far position. The value -1 defines that the eye is almost not being tracked due to extreme lower/left/near

position. If the eye isn't tracked at all the validity flag goes to 0 and all values for the represented eye will be set to 0.

Data Fields

double	positionRating-X	horizontal rating [-1; +1]
double	positionRating-Y	vertical rating [-1; +1]
double	positionRating-Z	distance rating [-1; +1]
double	relative-PositionX	horizontal position [-1; +1]
double	relative-PositionY	vertical position [-1; +1]
double	relative-PositionZ	depth/distance position [-1; +1]
int	validity	confidence of position and rating values [0; 1]

struct GazeChannelQualityStruct

This struct provides information about the last validation. Therefore a valid validation must be successfully completed before the [GazeChannelQualityStruct](#) can be updated. To update information in [GazeChannelQualityStruct](#) use function [iV_GetGazeChannelQuality](#).

Data Fields

double	gazeChannel-Quality-Binocular	Quality index of the averaged gaze channel. It lies in the range [0,1]. A value > 0.5 means it is acceptable. A NaN value means it is not provided.
double	gazeChannel-QualityLeft	Quality index of the left gaze channel. It lies in the range [0,1]. A value > 0.5 means it is acceptable. A NaN value means it is not provided.
double	gazeChannel-QualityRight	Quality index of the right gaze channel. It lies in the range [0,1]. A value > 0.5 means it is acceptable. A NaN value means it is not provided.

struct ImageStruct

Use this struct to get raw eye image, raw scene video image, or raw tracking monitor image. For receiving raw eye image (format: monochrome 8bpp) use [iV_GetEyeImage](#), or set the eye image callback with [iV_SetEyeImageCallback](#). For receiving raw scene video image (format: RGB 24bpp) use [iV_GetSceneVideo](#), or set the scene video callback with [iV_SetSceneVideoCallback](#). For receiving raw tracking monitor image (format: RGB 24bpp) use [iV_GetTrackingMonitor](#), or set the tracking monitor callback

with [iV_SetTrackingMonitorCallback](#).

Data Fields

char *	imageBuffer	pointer to image data
int	imageHeight	vertical size of the image [pixel]
int	imageSize	image data size [byte]
int	imageWidth	horizontal size of the image [pixel]

struct REDGeometryStruct

Use this struct to customize the RED geometry. See chapter [Setting up RED Geometry](#) in the iView X SDK Manual for details. For setting up the RED geometry parameters with [REDGeometryStruct](#) use function [iV_SetREDGeometry](#).

Data Fields

int	monitorSize	monitor size [inch] can be set to 19 or 22 used if redGeometry is set to monitorIntegrated only
enum RED-Geometry-Enum	redGeometry	defines which parameter is used.
int	redHeightOverFloor	distance floor to RED [mm] used if redGeometry is set to standalone only
int	redInclAngle	RED or RED-m inclination angle [degree] used if redGeometry is set to standalone only
int	redStimDist	distance RED to stimulus screen [mm] used if redGeometry is set to standalone only
int	redStimDist-Depth	horizontal distance RED-m to stimulus screen [mm] used if redGeometry is set to standalone only
int	redStimDist-Height	vertical distance RED-m to stimulus screen [mm] used if redGeometry is set to standalone only
char	setupName	name of the profile used if redGeometry is set to standalone only
int	stimHeight-OverFloor	distance floor to stimulus screen [mm] used if redGeometry is set to standalone only
int	stimX	horizontal stimulus calibration size [mm] used if redGeometry is set to standalone only
int	stimY	vertical stimulus calibration size [mm] used if redGeometry is set to standalone only

struct SampleStruct

This struct provides information about an eye data sample. To update information in [SampleStruct](#) use the function [iV_GetSample](#) or set the sample callback with [iV_SetSampleCallback](#).

Data Fields

struct EyeDataStruct	leftEye	stores information of the left eye (see EyeDataStruct for more information)
int	planeNumber	plane number of gaze data sample (only for HED HT)
struct EyeDataStruct	rightEye	stores information of the right eye (see EyeDataStruct for more information)
long long	timestamp	timestamp of current gaze data sample [microseconds]

struct SampleStruct32

This struct provides information about a eye data samples. To update information in [SampleStruct32](#) use the function [iV_GetSample32](#). The difference to [SampleStruct](#) is that the timestamp will be stored in milliseconds instead of microseconds.

Data Fields

struct EyeDataStruct	leftEye	stores information of the left eye (see EyeDataStruct for more information)
int	planeNumber	plane number of gaze data sample
struct EyeDataStruct	rightEye	stores information of the right eye (see EyeDataStruct for more information)
double	timestamp	timestamp of current gaze data sample [milliseconds]

struct SpeedModeStruct

This struct provides information about the speed modes used and supported by the connected iView eye tracking server. To update information in [SpeedModeStruct](#) use function [iV_GetSpeedModes](#).

Data Fields

int	numberOf-SpeedModes	number of supported speed modes
int	speedMode	the current speed mode
int	speedModes	an array of speed modes supported by the connected iView eye tracking server
int	version	version of the current data structure

struct SystemInfoStruct

This struct provides information about the iView eye tracking server version and the API version in use. To update data in [SystemInfoStruct](#) use the function [iV_GetSystemInfo](#).

Data Fields

int	API_- Buildnumber	build number of iView X SDK in use
int	API_Major- Version	major version number of iView X SDK in use
int	API_Minor- Version	minor version number of iView X SDK in use
int	iV_- Buildnumber	build number of iView eye tracking server in use
enum ETDevice	iV_ETDevice	type of eye tracking device
int	iV_Major- Version	major version number of iView eye tracking server in use
int	iV_Minor- Version	minor version number of iView eye tracking server in use
int	samplerate	sample rate of eye tracking device in use

struct TrackingStatusStruct

This struct provides information about the relative eye ball position within the tracking box. The information will be provided for each eye individually as well as for the geographical center between both eyes. To update information in [TrackingStatusStruct](#) use the function [iV_GetTrackingStatus](#).

Data Fields

struct Eye- PositionStruct	leftEye	stores information of the left eye (see EyePositionStruct for more information)
struct Eye- PositionStruct	rightEye	stores information of the right eye (see EyePositionStruct for more information)
long long	timestamp	timestamp of current tracking status sample [microseconds]
struct Eye- PositionStruct	total	stores information of the geometric average of both eyes (see Eye- PositionStruct for more information)

2.3 Callback Function Types

Typedefs

- typedef int(* [pDLLSetAOIHit](#))(int digitalOutoutValue)
- typedef int(* [pDLLSetCalibrationPoint](#))(struct [CalibrationPointStruct](#) calibrationPoint)
- typedef int(* [pDLLSetEvent](#))(struct [EventStruct](#) eventDataSample)
- typedef int(* [pDLLSetEyeImage](#))(struct [ImageStruct](#) eyeImage)
- typedef int(* [pDLLSetSample](#))(struct [SampleStruct](#) rawDataSample)
- typedef int(* [pDLLSetSceneVideo](#))(struct [ImageStruct](#) sceneVideo)
- typedef int(* [pDLLSetTrackingMonitor](#))(struct [ImageStruct](#) trackingMonitor)

Detailed Description

2.4 Functions

Functions

- `int iV_AbortCalibration ()`
- `int iV_AbortCalibrationPoint ()`
- `int iV_AcceptCalibrationPoint ()`
- `int iV_Calibrate ()`
- `int iV_ChangeCalibrationPoint (int number, int positionX, int positionY)`
- `int iV_ClearAOI ()`
- `int iV_ClearRecordingBuffer ()`
- `int iV_ConfigureFilter (enum FilterType filter, enum FilterAction action, void *data)`
- `int iV_Connect (char *sendIPAddress, int sendPort, char *recvIPAddress, int receivePort)`
- `int iV_ConnectLocal ()`
- `int iV_ContinueEyetracking ()`
- `int iV_ContinueRecording (char *etMessage)`
- `int iV_DefineAOI (struct AOIStruct *aoiData)`
- `int iV_DefineAOIPort (int port)`
- `int iV_DeleteREDGeometry (char *setupName)`
- `int iV_DisableAOI (char *aoiName)`
- `int iV_DisableAOIGroup (char *aoiGroup)`
- `int iV_DisableGazeDataFilter ()`
- `int iV_DisableProcessorHighPerformanceMode ()`
- `int iV_Disconnect ()`
- `int iV_EnableAOI (char *aoiName)`
- `int iV_EnableAOIGroup (char *aoiGroup)`
- `int iV_EnableGazeDataFilter ()`
- `int iV_EnableProcessorHighPerformanceMode ()`
- `int iV_GetAccuracy (struct AccuracyStruct *accuracyData, int visualization)`
- `int iV_GetAccuracyImage (struct ImageStruct *imageData)`
- `int iV_GetAOIOutputValue (int *aoiOutputValue)`
- `int iV_GetAvailableLptPorts (char *buffer, int *bufferSize)`
- `int iV_GetCalibrationParameter (struct CalibrationStruct *calibrationData)`
- `int iV_GetCalibrationPoint (int calibrationPointNumber, struct CalibrationPointStruct *calibrationPoint)`
- `int iV_GetCalibrationQuality (int calibrationPointNumber, struct CalibrationPointQualityStruct *left, struct CalibrationPointQualityStruct *right)`
- `int iV_GetCalibrationQualityImage (struct ImageStruct *imageData)`
- `int iV_GetCalibrationStatus (enum CalibrationStatusEnum *calibrationStatus)`
- `int iV_GetCurrentCalibrationPoint (struct CalibrationPointStruct *currentCalibrationPoint)`
- `int iV_GetCurrentREDGeometry (struct REDGeometryStruct *redGeometry)`
- `int iV_GetCurrentTimestamp (long long *currentTimestamp)`
- `int iV_GetDeviceName (char deviceName[64])`

- int [iV_GetEvent](#) (struct [EventStruct](#) *eventDataSample)
- int [iV_GetEvent32](#) (struct [EventStruct32](#) *eventDataSample)
- int [iV_GetEyeImage](#) (struct [ImageStruct](#) *imageData)
- int [iV_GetFeatureKey](#) (long long *featureKey)
- int [iV_GetGazeChannelQuality](#) (struct [GazeChannelQualityStruct](#) *qualityData)
- int [iV_GetGeometryProfiles](#) (int maxSize, char *profileNames)
- int [iV_GetLicenseDueDate](#) (struct [DateStruct](#) *licenseDueDate)
- int [iV_GetREDGeometry](#) (char *profileName, struct [REDGeometryStruct](#) *redGeometry)
- int [iV_GetSample](#) (struct [SampleStruct](#) *rawDataSample)
- int [iV_GetSample32](#) (struct [SampleStruct32](#) *rawDataSample)
- int [iV_GetSceneVideo](#) (struct [ImageStruct](#) *imageData)
- int [iV_GetSerialNumber](#) (char serialNumber[64])
- int [iV_GetSpeedModes](#) (struct [SpeedModeStruct](#) *speedModes)
- int [iV_GetSystemInfo](#) (struct [SystemInfoStruct](#) *systemInfoData)
- int [iV_GetTrackingMonitor](#) (struct [ImageStruct](#) *imageData)
- int [iV_GetTrackingStatus](#) (struct [TrackingStatusStruct](#) *trackingStatus)
- int [iV_GetUseCalibrationKeys](#) (int *enableKeys)
- int [iV_HideAccuracyMonitor](#) ()
- int [iV_HideEyeImageMonitor](#) ()
- int [iV_HideSceneVideoMonitor](#) ()
- int [iV_HideTrackingMonitor](#) ()
- int [iV_IsConnected](#) ()
- int [iV_LoadCalibration](#) (char *name)
- int [iV_Log](#) (char *logMessage)
- int [iV_PauseEyetracking](#) ()
- int [iV_PauseRecording](#) ()
- int [iV_Quit](#) ()
- int [iV_RecalibrateOnePoint](#) (int number)
- int [iV_ReleaseAOIPort](#) ()
- int [iV_RemoveAOI](#) (char *aoiName)
- int [iV_ResetCalibrationPoints](#) ()
- int [iV_SaveCalibration](#) (char *name)
- int [iV_SaveData](#) (char *filename, char *description, char *user, int overwrite)
- int [iV_SelectREDGeometry](#) (char *profileName)
- int [iV_SendCommand](#) (char *etMessage)
- int [iV_SendImageMessage](#) (char *etMessage)
- int [iV_SetAOIHitCallback](#) (pDLLSetAOIHit pAOIHitCallbackFunction)
- int [iV_SetCalibrationCallback](#) (pDLLSetCalibrationPoint pCalibrationCallbackFunction)
- int [iV_SetConnectionTimeout](#) (int time)
- int [iV_SetEventCallback](#) (pDLLSetEvent pEventCallbackFunction)
- int [iV_SetEventDetectionParameter](#) (int minDuration, int maxDispersion)
- int [iV_SetEyeImageCallback](#) (pDLLSetEyeImage pEyeImageCallbackFunction)
- int [iV_SetLicense](#) (const char *licenseKey)

- `int iV_SetLogger (int logLevel, char *filename)`
- `int iV_SetREDGeometry (struct REDGeometryStruct *redGeometry)`
- `int iV_SetResolution (int stimulusWidth, int stimulusHeight)`
- `int iV_SetSampleCallback (pDLLSetSample pSampleCallbackFunction)`
- `int iV_SetSceneVideoCallback (pDLLSetSceneVideo pSceneVideoCallbackFunction)`
- `int iV_SetSpeedMode (int speedMode)`
- `int iV_SetTrackingMonitorCallback (pDLLSetTrackingMonitor pTrackingMonitorCallbackFunction)`
- `int iV_SetTrackingParameter (int ET_PARAM_EYE, int ET_PARAM, int value)`
- `int iV_SetupCalibration (struct CalibrationStruct *calibrationData)`
- `int iV_SetupDebugMode (int enableDebugMode)`
- `int iV_SetupLptRecording (const char *portName, int enableRecording)`
- `int iV_SetUseCalibrationKeys (int enableKeys)`
- `int iV_ShowAccuracyMonitor ()`
- `int iV_ShowEyeImageMonitor ()`
- `int iV_ShowSceneVideoMonitor ()`
- `int iV_ShowTrackingMonitor ()`
- `int iV_Start (enum ETApplication etApplication)`
- `int iV_StartRecording ()`
- `int iV_StopRecording ()`
- `int iV_TestTTL (int value)`
- `int iV_Validate ()`

Detailed Description

Function Documentation

`int iV_AbortCalibration ()`

Aborts a calibration or validation if one is in progress. If the calibration or validation function is visualizing the calibration area the `iV_Calibrate` or `iV_Validate` function will return with `RET_CALIBRATION_ABORTED`. See also `iV_AbortCalibration`, `iV_AbortCalibrationPoint`, `iV_AcceptCalibrationPoint`, `iV_Calibrate`, `iV_ChangeCalibrationPoint`, `iV_GetCalibrationParameter`, `iV_GetCalibrationPoint`, `iV_GetCalibrationQuality`, `iV_GetCalibrationQualityImage`, `iV_GetCalibrationStatus`, `iV_GetCurrentCalibrationPoint`, `iV_GetUseCalibrationKeys`, `iV_LoadCalibration`, `iV_RecalibrateOnePoint`, `iV_ResetCalibrationPoints`, `iV_SaveCalibration`, `iV_SetCalibrationCallback`, `iV_SetResolution`, `iV_SetupCalibration`, `iV_SetUseCalibrationKeys`.

Return values

<code>RET_SUCCESS</code>	intended functionality has been fulfilled
<code>ERR_NOT_CONNECTED</code>	no connection established
<code>RET_NO_VALID_DATA</code>	no data available
<code>ERR_WRONG_DEVICE</code>	eye tracking device required for this function is not connected

int iV_AbortCalibrationPoint ()

Abort waiting for fixation for a calibration or validation point when the calibration or validation is in progress. If the latest calibration point has been accepted by the iView eye tracking server, the acceptance will be undone and the point unused. This allows the clients to customize the controlling logic of the calibration workflow, esp. when the calibration UI is implemented by user.

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_AcceptCalibrationPoint ()

Accepts a calibration or validation point if the calibration or validation is in progress. The participant needs to be tracked and has to fixate the calibration or validation point. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>RET_NO_VALID_DATA</i>	no data available
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_Calibrate ()

Starts a calibration procedure. To proceed, the participant needs to be tracked and has to fixate the calibration point. Depending on the calibration settings (which can be changed using [iV_SetupCalibration](#)) the user can accept the calibration points manually (by pressing SPACE or calling [iV_AcceptCalibrationPoint](#)) or abort the calibration (by pressing ESC or calling [iV_AbortCalibration](#))

If the calibration is visualized by the API ([CalibrationStruct::visualization](#) is set to 1) the function won't return until the calibration has been finished (closed automatically) or aborted (ESC).

If the [CalibrationStruct::visualization](#) is set to 0, [iV_Calibrate](#) returns immediately. The user has to care about the visualization of calibration points. Information about the current calibration point can be retrieved with [iV_GetCurrentCalibrationPoint](#) or with setting up the calibration callback using [iV_SetCalibrationCallback](#).

See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_WRONG_CALIBRATION_METHOD</i>	eye tracking device required for this calibration method is not connected

int iV_ChangeCalibrationPoint (int number, int positionX, int positionY)

Changes the position of a calibration point. This has to be done before the calibration process is started. The parameter number refers to the calibration method used. If this function is used with a RED or RED-m device, the change is applied to the currently selected profile. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Parameters

<i>number</i>	selected calibration point
<i>positionX</i>	new X position on screen [pixel]
<i>positionY</i>	new Y position on screen [pixel]

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_NO_RESPONSE_FROM_IVIEWX</i>	no response from iView X; check calibration name / identifier

int iV_ClearAOI ()

Removes all trigger AOIs. See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#),

[iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_AOI_ACCESS</i>	failed to access AOI data

[int iV_ClearRecordingBuffer \(\)](#)

Clears the recorded data buffer. The recording buffer needs to be stopped using "iV_StopRecording" before it can be cleared. If you are using an "HED", the scene video buffer is cleared, too. See also [iV_ClearRecordingBuffer](#), [iV_ContinueRecording](#), [iV_PauseRecording](#), [iV_SaveData](#), [iV_SendImage-Message](#), [iV_StartRecording](#), [iV_StopRecording](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_EMPTY_DATA_BUFFER</i>	recording buffer is empty
<i>ERR_RECORDING_DATA_BUFFER</i>	recording is activated
<i>ERR_PAUSED_DATA_BUFFER</i>	recording is in pause state
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

[int iV_ConfigureFilter \(enum FilterType filter, enum FilterAction action, void * data \)](#)

Queries or sets filter parameters. The usage of the parameter data depends on the parameter action,.

Parameters

<i>filter</i>	filter type that is configured. See FilterType
<i>action</i>	type of action. See FilterAction
<i>data</i>	A void pointer that can be casted to a data type depending on filter type. Please refer to FilterType for details. Content of the parameter depends on filter action, see FilterType . FilterAction::Query: data is filled with current filter settings. FilterAction::Set: data is passed to configure the filter

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_Connect (char * sendIPAddress, int sendPort, char * recvIPAddress, int receivePort)

Establishes a connection to the iView eye tracking server. **iV_Connect** will not return until a connection has been established. If no connection can be established, the function will return after the time span defined by **iV_SetConnectionTimeout**. Default time span is 3 seconds. For systems running with S-MI iViewRED 4.2 or higher, it is no longer required to define the parameters **recvIPAddress** and **receivePort**. See also **iV_Connect**, **iV_ConnectLocal**, **iV_ContinueEyetracking**, **iV_Disconnect**, **iV_GetDeviceName**, **iV_GetFeatureKey**, **iV_GetLicenseDueDate**, **iV_GetSerialNumber**, **iV_GetSpeedModes**, **iV_GetSystemInfo**, **iV_IsConnected**, **iV_PauseEyetracking**, **iV_Quit**, **iV_SetConnectionTimeout**, **iV_SetLicense**, **iV_SetSpeedMode**, **iV_Start**.

Parameters

sendIPAddress	IP address of iView X computer
sendPort	port being used by iView X SDK for sending data to iView X
recvIPAddress	IP address of local computer
receivePort	port being used by iView X SDK for receiving data from iView X

Return values

RET_SUCCESS	intended functionality has been fulfilled
ERR_SERVER_NOT_FOUND	no iView eye tracking server detected
ERR_EYETRACKING_APPLICATION_NOT_RUNNING	no eye tracking application running
ERR_WRONG_PARAMETER	parameter out of range
ERR_COULD_NOT_CONNECT	failed to establish connection

int iV_ConnectLocal ()

Establishes a connection to the iView eye tracking server. **iV_ConnectLocal** will not return until a connection has been established. If no connection can be established the function will return after the time span defined by **iV_SetConnectionTimeout**. Default time span is 3 seconds.

iV_ConnectLocal can only connect with RED-m or RED-OEM devices connected to the same PC. See also **iV_Connect**, **iV_ConnectLocal**, **iV_ContinueEyetracking**, **iV_Disconnect**, **iV_GetDeviceName**, **iV_GetFeatureKey**, **iV_GetLicenseDueDate**, **iV_GetSerialNumber**, **iV_GetSpeedModes**, **iV_GetSystemInfo**, **iV_IsConnected**, **iV_PauseEyetracking**, **iV_Quit**, **iV_SetConnectionTimeout**, **iV_SetLicense**, **iV_SetSpeedMode**, **iV_Start**.

Return values

RET_SUCCESS	intended functionality has been fulfilled
ERR_SERVER_NOT_FOUND	no iView eye tracking server detected
ERR_EYETRACKING_APPLICATION_NOT_RUNNING	no eye tracking application running
ERR_COULD_NOT_CONNECT	failed to establish connection

int iV_ContinueEyetracking ()

Wakes up and enables the eye tracking application from suspend mode to continue processing gaze data. The application can be set to suspend mode by calling [iV_PauseEyetracking](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_ContinueRecording (char * etMessage)

Continues gaze data recording. A HED video recording can neither be paused nor continued. [iV_ContinueRecording](#) does not return until gaze recording is continued. Before it can be continued, the data needs to be paused using [iV_PauseRecording](#). Additionally this function allows a message to be stored inside the idf data buffer. See also [iV_ClearRecordingBuffer](#), [iV_ContinueRecording](#), [iV_PauseRecording](#), [iV_SaveData](#), [iV_SendImageMessage](#), [iV_StartRecording](#), [iV_StopRecording](#).

Parameters

<i>etMessage</i>	text message that will be written to data file
------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_EMPTY_DATA_BUFFER</i>	recording buffer is empty
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_DefineAOI (struct AOIStruct * aoIData)

Defines an AOI. The API can handle up to 20 AOIs. See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#), [iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#).

Parameters

<i>aoIData</i>	see reference information for AOIStruct
----------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_DefineAOIPort (int port)

Selects a port for sending out TTL trigger. See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#), [iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#).

Parameters

<i>port</i>	port address
-------------	--------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_COULD_NOT_OPEN_PORT</i>	failed to open port

int iV_DeleteREDGeometry (char * setupName)

Deletes the RED-m geometry setup with the given name. It is not possible to delete a geometry profile if it is currently in use. See chapter [Setting up RED Geometry](#) in the iView X SDK Manual.

Parameters

<i>setupName</i>	name of the geometry setup which will be deleted
------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_DisableAOI (char * aoIName)

Disables all AOIs with the given name. See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#), [iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#).

Parameters

<i>aoIName</i>	name of the AOI which will be disabled
----------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no data available
<i>ERR_AOI_ACCESS</i>	failed to access AOI data
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_DisableAOIGroup (char * aoigroup)

Disables an AOI group. See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#), [iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#).

Parameters

<i>aoigroup</i>	name of the AOI group which will be disabled
-----------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no data available
<i>ERR_AOI_ACCESS</i>	failed to access AOI data

int iV_DisableGazeDataFilter ()

Disables the raw data filter. The gaze data filter can be enabled using [iV_EnableGazeDataFilter](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_DisableProcessorHighPerformanceMode ()

Disables a CPU high performance mode allowing the CPU to reduce the performance. See also [iV_EnableProcessorHighPerformanceMode](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_Disconnect ()

Disconnects from iView eye tracking server. **iV_Disconnect** will not return until the connection has been disconnected. After this function has been called no other function or device can communicate with iView eye tracking server. See also **iV_Connect**, **iV_ConnectLocal**, **iV_ContinueEyetracking**, **iV_Disconnect**, **iV_GetDeviceName**, **iV_GetFeatureKey**, **iV_GetLicenseDueDate**, **iV_GetSerialNumber**, **iV_GetSpeedModes**, **iV_GetSystemInfo**, **iV_IsConnected**, **iV_PauseEyetracking**, **iV_Quit**, **iV_SetConnectionTimeout**, **iV_SetLicense**, **iV_SetSpeedMode**, **iV_Start**.

Return values

RET_SUCCESS	intended functionality has been fulfilled
ERR_DELETE_SOCKET	failed to delete sockets

int iV_EnableAOI (char * aoIName)

Enables all AOIs with the given name. See also **iV_ClearAOI**, **iV_DefineAOI**, **iV_DefineAOIPort**, **iV_DisableAOI**, **iV_DisableAOIGroup**, **iV_EnableAOI**, **iV_EnableAOIGroup**, **iV_GetAOIOutputValue**, **iV_ReleaseAOIPort**, **iV_RemoveAOI**, **iV_SetAOIHitCallback**, **iV_TestTTL**.

Parameters

aoIName	name of the AOI which will be enabled
----------------	---------------------------------------

Return values

RET_SUCCESS	intended functionality has been fulfilled
RET_NO_VALID_DATA	no data available
ERR_AOI_ACCESS	failed to access AOI data

int iV_EnableAOIGroup (char * aoIGroup)

Enables an AOI group. See also **iV_ClearAOI**, **iV_DefineAOI**, **iV_DefineAOIPort**, **iV_DisableAOI**, **iV_DisableAOIGroup**, **iV_EnableAOI**, **iV_EnableAOIGroup**, **iV_GetAOIOutputValue**, **iV_ReleaseAOIPort**, **iV_RemoveAOI**, **iV_SetAOIHitCallback**, **iV_TestTTL**.

Parameters

aoIGroup	name of the AOI group which will be enabled
-----------------	---

Return values

RET_SUCCESS	intended functionality has been fulfilled
RET_NO_VALID_DATA	no data available
ERR_AOI_ACCESS	failed to access AOI data

int iV_EnableGazeDataFilter ()

Enables a gaze data filter. This API bilateral filter was implemented due to special HCI application requirements. The gaze data filter can be disabled using [iV_DisableGazeDataFilter](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
--------------------	---

int iV_EnableProcessorHighPerformanceMode ()

Enables a CPU high performance mode to prevent the CPU from reducing the performance. See also [iV_DisableProcessorHighPerformanceMode](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_GetAccuracy (struct AccuracyStruct * accuracyData, int visualization)

Updates [AccuracyStruct](#) accuracyData with validated accuracy results. Before accuracy data is accessible the accuracy needs to be validated with [iV_Validate](#). If the parameter `visualization` is set to 1 the accuracy data will be visualized in a dialog window. See also [iV_GetAccuracy](#), [iV_GetAccuracyImage](#), [iV_GetGazeChannelQuality](#), [iV_HideAccuracyMonitor](#), [iV_ShowAccuracyMonitor](#), [iV_Validate](#) and the chapter [Validation](#) in the iView X SDK Manual.

Parameters

<i>accuracyData</i>	see reference information for AccuracyStruct
<i>visualization</i>	0: no visualization 1: accuracy data will be visualized in a dialog window

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_NOT_CALIBRATED</i>	system is not calibrated
<i>ERR_NOT_VALIDATED</i>	system is not validated
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_GetAccuracyImage (struct ImageStruct * *imageData*)

Updates *imageData* struct with drawn accuracy results. Before accuracy data is accessible the accuracy needs to be validated with [iV_Validate](#). See also [iV_GetAccuracy](#), [iV_GetAccuracyImage](#), [iV_GetGazeChannelQuality](#), [iV_HideAccuracyMonitor](#), [iV_ShowAccuracyMonitor](#), [iV_Validate](#) and the chapter [Validation](#) in the iView X SDK Manual.

Parameters

<i>imageData</i>	see reference information for ImageStruct
------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_NOT_CALIBRATED</i>	system is not calibrated
<i>ERR_NOT_VALIDATED</i>	system is not validated

int iV_GetAOIOutputValue (int * *aoiOutputValue*)

Gives back the AOI value See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#), [iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#).

Parameters

<i>aoiOutputValue</i>	provides the AOI output value
-----------------------	-------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_AOI_NOT_DEFINED</i>	no defined AOI found

int iV_GetAvailableLptPorts (char * *buffer*, int * *bufferSize*)

Retrieves the available LPT ports from the computer running the eye tracking server which can be used for LPT data recording. The caller has to supply a sufficiently large string buffer to the function. The buffer is filled with a string list containing the names of the available LPT port, separated by a semi colon (;). An empty string indicates that there are no suitable LPT ports available.

Parameters

<i>buffer</i>	The buffer that is filled with a list of available LPT port names, separated by a semi colon (;)
---------------	--

<i>bufferSize</i>	pointer to a variable that contains the size of the buffer in bytes; when the function returns with RET_SUCCESS, the variable will contain the size of the used buffer, including the terminating '\0' character
-------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_INSUFFICIENT_BUFFER_SIZE</i>	the provided buffer is too small and has not been altered
<i>ERR_WRONG_PARAMETER</i>	either the buffer or bufferSize parameter are NULL
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_GetCalibrationParameter (struct CalibrationStruct * calibrationData)

Updates stored *calibrationData* information with currently selected parameters. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Parameters

<i>calibrationData</i>	see reference information for CalibrationStruct
------------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_GetCalibrationPoint (int calibrationPointNumber, struct CalibrationPointStruct * calibrationPoint)

Delivers information about a calibration point. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Parameters

<i>calibration-PointNumber</i>	number of requested calibration point
<i>calibrationPoint</i>	information of requested calibration point, stored in CalibrationPointStruct

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_GetCalibrationQuality (int *calibrationPointNumber*, struct **CalibrationPointQualityStruct * *left*, struct **CalibrationPointQualityStruct** * *right*)**

Delivers fixation quality information about a calibration point. If the output data holder is NULL, simply no data will be returned.

Parameters

<i>calibration-PointNumber</i>	number of requested calibration point
<i>left</i>	requested quality information for the left eye, stored in CalibrationPointQualityStruct
<i>right</i>	requested quality information for the right eye, stored in CalibrationPointQualityStruct

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_GetCalibrationQualityImage (struct **ImageStruct * *imageData*)**

Updates *imageData* struct with drawn calibration quality data. Calibration is required before calling this function.

Parameters

<i>imageData</i>	see reference information for ImageStruct
------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_NOT_CALIBRATED</i>	system is not calibrated

int iV_GetCalibrationStatus (enum **CalibrationStatusEnum * *calibrationStatus*)**

Updates *calibrationStatus* information. The client needs to be connected to the iView eye tracking server. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#),

[iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOne-Point](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Parameters

<i>calibration-Status</i>	see reference information for CalibrationStatusEnum
---------------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_DATA_INVALID</i>	no new data available
<i>ERR_CONNECTION_NOT_ESTABLISHED</i>	no connection established

[int iV_GetCurrentCalibrationPoint \(struct \[CalibrationPointStruct\]\(#\) * *currentCalibrationPoint* \)](#)

Updates data in [CalibrationPointStruct](#) *currentCalibrationPoint* with current calibration point data. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Parameters

<i>current-Calibration-Point</i>	information of requested calibration point, stored in CalibrationPointStruct
----------------------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established

[int iV_GetCurrentREDGeometry \(struct \[REDGeometryStruct\]\(#\) * *redGeometry* \)](#)

Gets the currently loaded RED geometry. See chapter [Setting up RED Geometry](#) in the iView X SDK Manual and [iV_DeleteREDGeometry](#), [iV_GetCurrentREDGeometry](#), [iV_GetGeometryProfiles](#), [iV_GetREDGeometry](#), [iV_SelectREDGeometry](#), [iV_SetREDGeometry](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_CONNECTION_NOT_ESTABLISHED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_GetCurrentTimestamp (long long * *currentTimestamp*)

Provides the current eye tracker timestamp in microseconds. See also [iV_GetCurrentTimestamp](#), [iV_GetEvent](#), [iV_GetEvent32](#), [iV_GetSample](#), [iV_GetSample32](#), [iV_GetTrackingStatus](#), [iV_SetEventCallback](#), [iV_SetEventDetectionParameter](#), [iV_SetSampleCallback](#).

Parameters

<i>currentTimestamp</i>	information of requested time stamp
-------------------------	-------------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_GetDeviceName (char *deviceName*[64])

Updated the device name information of the connected device.

Parameters

<i>deviceName</i>	the name of the requested device
-------------------	----------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_GetEvent (struct EventStruct * eventDataSample)

Updates data from [EventStruct](#) eventDataSample with current event data. See also [iV_GetCurrent-Timestamp](#), [iV_GetEvent](#), [iV_GetEvent32](#), [iV_GetSample](#), [iV_GetSample32](#), [iV_GetTrackingStatus](#), [iV_SetEventCallback](#), [iV_SetEventDetectionParameter](#), [iV_SetSampleCallback](#).

Parameters

<i>eventData-Sample</i>	see reference information for EventStruct
-------------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_GetEvent32 (struct EventStruct32 * eventDataSample)

Updates data from [EventStruct32](#) eventDataSample with current event data. See also [iV_GetCurrent-Timestamp](#), [iV_GetEvent](#), [iV_GetEvent32](#), [iV_GetSample](#), [iV_GetSample32](#), [iV_GetTrackingStatus](#), [iV_SetEventCallback](#), [iV_SetEventDetectionParameter](#), [iV_SetSampleCallback](#).

Parameters

<i>eventData-Sample</i>	see reference information for EventStruct32
-------------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_GetEyeImage (struct ImageStruct * imageData)

Updates imageData with current eye image.

Parameters

<i>imageData</i>	see reference information for ImageStruct
------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_GetFeatureKey (long long * featureKey)

Gets the device specific feature key. Used for RED-OEM devices only.

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_GetGazeChannelQuality (struct GazeChannelQualityStruct * qualityData)

Retrieve gaze quality data. Fills [GazeChannelQualityStruct](#) qualityData with validated accuracy results. Before quality data is accessible the system needs to be validated with [iV_Validate](#). See also [iV_GetAccuracy](#), [iV_GetAccuracyImage](#), [iV_GetGazeChannelQuality](#), [iV_HideAccuracyMonitor](#), [iV_ShowAccuracyMonitor](#), [iV_Validate](#) and the chapter [Validation](#) in the iView X SDK Manual.

Parameters

<i>qualityData</i>	see reference information for GazeChannelQualityStruct
--------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_NOT_CALIBRATED</i>	system is not calibrated
<i>ERR_NOT_VALIDATED</i>	system is not validated
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_GetGeometryProfiles (int maxSize, char * profileNames)

Gets all available profiles by name. They will be written comma-separated in the char buffer. The user needs to be sure that the buffer is not smaller than the needed buffer length.

See chapter [Setting up RED Geometry](#) in the iView X SDK Manual and [iV_DeleteREDGeometry](#), [iV_GetCurrentREDGeometry](#), [iV_GetGeometryProfiles](#), [iV_GetREDGeometry](#), [iV_SelectREDGeometry](#), [iV_SetREDGeometry](#).

Parameters

<i>maxSize</i>	the length of the string profileNames
<i>profileNames</i>	an empty string where profile names will be put in

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_GetLicenseDueDate (struct DateStruct * *licenseDueDate*)

Gets the system license expiration date. The license will not expire if the license is set to 00.00.0000. See also [iV_Connect](#), [iV_ConnectLocal](#), [iV_ContinueEyetracking](#), [iV_Disconnect](#), [iV_GetDeviceName](#), [iV_GetFeatureKey](#), [iV_GetLicenseDueDate](#), [iV_GetSerialNumber](#), [iV_GetSpeedModes](#), [iV_GetSystemInfo](#), [iV_IsConnected](#), [iV_PauseEyetracking](#), [iV_Quit](#), [iV_SetConnectionTimeout](#), [iV_SetLicense](#), [iV_SetSpeedMode](#), [iV_Start](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_GetREDGeometry (char * *profileName*, struct REDGeometryStruct * *redGeometry*)

Gets the geometry data of a requested profile without selecting them. See chapter [Setting up RED Geometry](#) in the iView X SDK Manual and [iV_DeleteREDGeometry](#), [iV_GetCurrentREDGeometry](#), [iV_GetGeometryProfiles](#), [iV_GetREDGeometry](#), [iV_SelectREDGeometry](#), [iV_SetREDGeometry](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_GetSample (struct SampleStruct * rawDataSample)

Updates data in [SampleStruct](#) `rawDataSample` with current eye tracking data. See also [iV_GetCurrentTimestamp](#), [iV_GetEvent](#), [iV_GetEvent32](#), [iV_GetSample](#), [iV_GetSample32](#), [iV_GetTrackingStatus](#), [iV_SetEventCallback](#), [iV_SetEventDetectionParameter](#), [iV_SetSampleCallback](#).

Parameters

<i>rawData-Sample</i>	see reference information for SampleStruct
-----------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_GetSample32 (struct SampleStruct32 * rawDataSample)

Updates data in [SampleStruct32](#) `rawDataSample` with current eye tracking data sample. See also [iV_GetCurrentTimestamp](#), [iV_GetEvent](#), [iV_GetEvent32](#), [iV_GetSample](#), [iV_GetSample32](#), [iV_GetTrackingStatus](#), [iV_SetEventCallback](#), [iV_SetEventDetectionParameter](#), [iV_SetSampleCallback](#).

Parameters

<i>rawData-Sample</i>	see reference information for SampleStruct32
-----------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_GetSceneVideo (struct ImageStruct * imageData)

Updates [ImageStruct](#) `imageData` with current scene video image. This functions is available for HED only.

Parameters

<i>imageData</i>	see reference information for ImageStruct
------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_GetSerialNumber (char serialNumber[64])

Retrieve the serial number information of the connected device. See also [iV_Connect](#), [iV_ConnectLocal](#), [iV_ContinueEyetracking](#), [iV_Disconnect](#), [iV_GetDeviceName](#), [iV_GetFeatureKey](#), [iV_GetLicenseDueDate](#), [iV_GetSerialNumber](#), [iV_GetSpeedModes](#), [iV_GetSystemInfo](#), [iV_IsConnected](#), [iV_PauseEyetracking](#), [iV_Quit](#), [iV_SetConnectionTimeout](#), [iV_SetLicense](#), [iV_SetSpeedMode](#), [iV_Start](#).

Parameters

<i>serialNumber</i>	the serial number of the requested device
---------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_WRONG_IVIEWX_VERSION</i>	wrong version of iView X

int iV_GetSpeedModes (struct SpeedModeStruct * speedModes)

This function retrieves the speed modes used and supported by the connected iView eye tracking server.

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMTER</i>	the requested speed mode is not supported by the connected device
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_GetSystemInfo (struct SystemInfoStruct * systemInfoData)

Query system information. [SystemInfoStruct](#) systemInfoData is updated with current system information. See also [iV_Connect](#), [iV_ConnectLocal](#), [iV_ContinueEyetracking](#), [iV_Disconnect](#), [iV_GetDeviceName](#), [iV_GetFeatureKey](#), [iV_GetLicenseDueDate](#), [iV_GetSerialNumber](#), [iV_GetSpeedModes](#), [iV_GetSystemInfo](#), [iV_IsConnected](#), [iV_PauseEyetracking](#), [iV_Quit](#), [iV_SetConnectionTimeout](#), [iV_SetLicense](#), [iV_SetSpeedMode](#), [iV_Start](#).

Parameters

<i>systemInfoData</i>	see reference information for SystemInfoStruct
-----------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no data available

int iV_GetTrackingMonitor (struct ImageStruct * imageData)

Updates [ImageStruct](#) imageData with current tracking monitor image.

Parameters

<i>imageData</i>	see reference information for ImageStruct
------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_GetTrackingStatus (struct TrackingStatusStruct * trackingStatus)

Updates [TrackingStatusStruct](#) trackingStatus with current tracking status.

Parameters

<i>trackingStatus</i>	see reference information for TrackingStatusStruct
-----------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no new data available
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_HideAccuracyMonitor ()

Hides accuracy monitor window which can be opened by [iV_ShowAccuracyMonitor](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_WINDOW_IS_CLOSED</i>	window is already closed
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_HideEyeImageMonitor ()

Hides eye image monitor window which can be opened by [iV_ShowEyeImageMonitor](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_WINDOW_IS_CLOSED</i>	window is already closed
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_HideSceneVideoMonitor ()

Hides scene video monitor window which can be opened by [iV_ShowSceneVideoMonitor](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_WINDOW_IS_CLOSED</i>	window is already closed
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_HideTrackingMonitor ()

Hides tracking monitor window which can be opened by [iV_ShowTrackingMonitor](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_WINDOW_IS_CLOSED</i>	window is already closed
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_IsConnected ()

Checks if connection to iView eye tracking server is still established. See also [iV_Connect](#), [iV_ConnectLocal](#), [iV_ContinueEyetracking](#), [iV_Disconnect](#), [iV_GetDeviceName](#), [iV_GetFeatureKey](#), [iV_GetLicenseDueDate](#), [iV_GetSerialNumber](#), [iV_GetSpeedModes](#), [iV_GetSystemInfo](#), [iV_IsConnected](#), [iV_PauseEyetracking](#), [iV_Quit](#), [iV_SetConnectionTimeout](#), [iV_SetLicense](#), [iV_SetSpeedMode](#), [iV_Start](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_LoadCalibration (char * name)

Loads a previously saved calibration. A calibration has to be saved by using [iV_SaveCalibration](#). See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Parameters

<i>name</i>	calibration name / identifier
-------------	-------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_IVIEWX_VERSION</i>	wrong version of iView X
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_NO_RESPONSE_FROM_IVIEWX</i>	no response from iView X; check calibration name / identifier

int iV_Log (char * logMessage)

Writes logMessage into log file.

Parameters

<i>logMessage</i>	message that shall be written to the log file
-------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_ACCESS_TO_FILE</i>	failed to access log file

int iV_PauseEyetracking ()

Suspend the eye tracking application and disables calculation of gaze data. The application can be reactivated by calling [iV_ContinueEyetracking](#). See also [iV_Connect](#), [iV_ConnectLocal](#), [iV_ContinueEyetracking](#), [iV_Disconnect](#), [iV_GetDeviceName](#), [iV_GetFeatureKey](#), [iV_GetLicenseDueDate](#), [iV_GetSerialNumber](#), [iV_GetSpeedModes](#), [iV_GetSystemInfo](#), [iV_IsConnected](#), [iV_PauseEyetracking](#), [iV_Quit](#), [iV_SetConnectionTimeout](#), [iV_SetLicense](#), [iV_SetSpeedMode](#), [iV_Start](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_PauseRecording ()

Pauses gaze data recording. A HED video recording can neither be paused nor continued. [iV_PauseRecording](#) does not return until gaze recording is paused. See also [iV_ClearRecordingBuffer](#), [iV_ContinueRecording](#), [iV_PauseRecording](#), [iV_SaveData](#), [iV_SendImageMessage](#), [iV_StartRecording](#), [iV_StopRecording](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_EMPTY_DATA_BUFFER</i>	recording buffer is empty
<i>ERR_FULL_DATA_BUFFER</i>	data buffer is full
<i>ERR_PAUSED_DATABUFFER</i>	already in pause state
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_Quit ()

Disconnects and closes iView eye tracking server. After this function has been called no other function or application can communicate with iView eye tracking server. See also [iV_Connect](#), [iV_ConnectLocal](#), [iV_ContinueEyetracking](#), [iV_Disconnect](#), [iV_GetDeviceName](#), [iV_GetFeatureKey](#), [iV_GetLicenseDueDate](#), [iV_GetSerialNumber](#), [iV_GetSpeedModes](#), [iV_GetSystemInfo](#), [iV_IsConnected](#), [iV_PauseEyetracking](#), [iV_Quit](#), [iV_SetConnectionTimeout](#), [iV_SetLicense](#), [iV_SetSpeedMode](#), [iV_Start](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_DELETE_SOCKET</i>	failed to delete sockets
<i>ERR_WRONG_IVIEWX_VERSION</i>	wrong version of iView X

int iV_RecalibrateOnePoint (int number)

Restarts a calibration procedure with a point from the latest calibration process. The point is specified by its index in the calibration point profile (counted from 1). If the requested point is not found, an error code will be returned. The number of calibration points can be retrieved via [iV_GetCalibrationQuality](#).

This function can be used to improve the final calibration quality in cases when some points from the previous calibration were missed unexpectedly. With this function you can re-include that point to your calibration.

This function follows the workflow of [iV_Calibrate](#) except that [iV_SetupCalibration](#) must not be called after the end of the last calibration and before calling this function.

If [CalibrationStruct::visualization](#) was set to 0, [iV_RecalibrateOnePoint](#) returns immediately. The user has to care about the visualization of calibration points. Information about the current calibration point can be retrieved with [iV_GetCurrentCalibrationPoint](#) or with setting up the calibration callback using [iV_SetCalibrationCallback](#).

See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_PARAMETER</i>	the point to recalibrate is not found
<i>ERR_NOT_CALIBRATED</i>	no previous calibration exists
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

<i>ERR_WRONG_CALIBRATION_METHOD</i>	eye tracking device required for this calibration method is not connected or the method chosen for recalibration is different to the previous calibration.
-------------------------------------	--

int iV_ReleaseAOIPort ()

Releases the port for sending TTL trigger. See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#), [iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_COULD_NOT_CLOSE_PORT</i>	failed to close TTL port

int iV_RemoveAOI (char * *aoiName*)

Removes all AOIs with the given name. See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#), [iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#).

Parameters

<i>aoiName</i>	name of the AOI which will be removed
----------------	---------------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no data available
<i>ERR_AOI_ACCESS</i>	failed to access AOI data

int iV_ResetCalibrationPoints ()

Resets all calibration points to its default position. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_SaveCalibration (char * name)

Saves a calibration with a custom name. To save a calibration it is required that a successful calibration already has been completed. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Parameters

<i>name</i>	calibration name / identifier
-------------	-------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_NOT_CALIBRATED</i>	system is not calibrated
<i>ERR_WRONG_IVIEWX_VERSION</i>	wrong version of iView X
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_SaveData (char * filename, char * description, char * user, int overwrite)

Writes recorded data buffer to disc. The data recording needs to be stopped using [iV_StopRecording](#) before the data buffer can be saved to given location. The `filename` can include the path. If the connected eye tracking device is a HED, scene video buffer is written, too. [iV_SaveData](#) will not return until the data has been saved.

Parameters

<i>filename</i>	full path including the filename of the data file being created
<i>description</i>	Optional experiment description tag stored in the idf file. This tag is available in BeGaze and in the text export from an idf file.
<i>user</i>	Optional name of test person. This tag is available in BeGaze and in the text export from an idf file.
<i>overwrite</i>	Overwriting policy. 0: do not overwrite file <code>filename</code> if it already exists 1: overwrite file <code>filename</code> if it already exists

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_EMPTY_DATA_BUFFER</i>	recording buffer is empty
<i>ERR_RECORDING_DATA_BUFFER</i>	recording is activated
<i>ERR_PAUSED_DATA_BUFFER</i>	recording is in pause state
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_SelectREDGeometry (char * *profileName*)

Selects a predefined geometry profile. See chapter [Setting up RED Geometry](#) in the iView X SDK Manual and [iV_DeleteREDGeometry](#), [iV_GetCurrentREDGeometry](#), [iV_GetGeometryProfiles](#), [iV_GetREDGeometry](#), [iV_SelectREDGeometry](#), [iV_SetREDGeometry](#).

Parameters

<i>profileName</i>	name of the selected profile which should be selected
--------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_FEATURE_NOT_LICENSED</i>	this feature was not licensed by the API user

int iV_SendCommand (char * *etMessage*)

Sends a remote command to iView eye tracking server. Please refer to the iView X help file for further information about remote commands. Important Note: This function is temporary and will not be supported in subsequent versions. See also [iV_ClearRecordingBuffer](#), [iV_ContinueRecording](#), [iV_PauseRecording](#), [iV_SaveData](#), [iV_SendImageMessage](#), [iV_StartRecording](#), [iV_StopRecording](#).

Parameters

<i>etMessage</i>	iView X remote command
------------------	------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_SendImageMessage (char * etMessage)

Sends a text message to iView X idf recording data file. If the *etMessage* has the suffix ".jpg", ".-bmp", ".png", or ".avi" BeGaze will separate the data buffer automatically into according trials. See also [iV_ClearRecordingBuffer](#), [iV_ContinueRecording](#), [iV_PauseRecording](#), [iV_SaveData](#), [iV_SendImageMessage](#), [iV_StartRecording](#), [iV_StopRecording](#).

Parameters

<i>etMessage</i>	Any text message to separate trials (image name containing extensions) or any idf data marker
------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_SetAOIHitCallback (pDLLSetAOIHit pAOIHitCallbackFunction)

Sets a callback function for the AOI hit functions. The function will be called if the iView eye tracking server has calculated an AOI hit. For usage of this function AOI's needs to be defined. See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#), [iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#) Important note: Algorithms with high processor usage and long calculation time shouldn't run within this callback due to a higher probability of data loss See also [iV_GetCurrentTimestamp](#), [iV_GetEvent](#), [iV_GetEvent32](#), [iV_GetSample](#), [iV_GetSample32](#), [iV_GetTrackingStatus](#), [iV_SetEventCallback](#), [iV_SetEventDetectionParameter](#), [iV_SetSampleCallback](#).

Parameters

<i>pAOIHit-Callback-Function</i>	pointer to AOIHitCallbackFunction
----------------------------------	-----------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_SetCalibrationCallback (pDLLSetCalibrationPoint pCalibrationCallbackFunction)

Sets a callback function for the calibration and validation process. The callback function will be called after a calibration or validation was started, after a calibration or validation point was accepted, or if

the calibration or validation was finished successfully or unsuccessfully. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Parameters

<i>pCalibration-Callback-Function</i>	pointer to CalibrationCallbackFunction
---------------------------------------	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_SetConnectionTimeout (int time)

Defines a customized timeout for how long [iV_Connect](#) tries to connect to iView eye tracking server. See also [iV_Connect](#), [iV_ConnectLocal](#), [iV_ContinueEyetracking](#), [iV_Disconnect](#), [iV_GetDeviceName](#), [iV_GetFeatureKey](#), [iV_GetLicenseDueDate](#), [iV_GetSerialNumber](#), [iV_GetSpeedModes](#), [iV_GetSystemInfo](#), [iV_IsConnected](#), [iV_PauseEyetracking](#), [iV_Quit](#), [iV_SetConnectionTimeout](#), [iV_SetLicense](#), [iV_SetSpeedMode](#), [iV_Start](#).

Parameters

<i>time</i>	the time [sec] iV_Connect is waiting for iView X response
-------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_SetEventCallback (pDLLSetEvent pEventCallbackFunction)

Sets a callback function for the event data. The function will be called if a real-time detected fixation has been started or ended. Important note: Algorithms with high processor usage and long calculation time shouldn't run within this callback due to a higher probability of data loss. See also [iV_GetCurrentTimestamp](#), [iV_GetEvent](#), [iV_GetEvent32](#), [iV_GetSample](#), [iV_GetSample32](#), [iV_GetTrackingStatus](#), [iV_SetEventCallback](#), [iV_SetEventDetectionParameter](#), [iV_SetSampleCallback](#).

Parameters

<i>pEvent-Callback-Function</i>	pointer to EventCallbackFunction
---------------------------------	----------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_SetEventDetectionParameter (int *minDuration*, int *maxDispersion*)

Defines the detection parameter for online fixation detection algorithm. See also [iV_GetCurrent-Timestamp](#), [iV_GetEvent](#), [iV_GetEvent32](#), [iV_GetSample](#), [iV_GetSample32](#), [iV_GetTrackingStatus](#), [iV_SetEventCallback](#), [iV_SetEventDetectionParameter](#), [iV_SetSampleCallback](#).

Parameters

<i>minDuration</i>	minimum fixation duration [ms]
<i>maxDispersion</i>	maximum dispersion [pixel] for head tracking systems or [deg] for non head tracking systems

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_SetEyeImageCallback (pDLLSetEyeImage *pEyeImageCallbackFunction*)

Sets a callback function for the eye image data. The function will be called if a new eye image is available. The image format is monochrome 8bpp. Important note: Algorithms with high processor usage and long calculation time shouldn't run within this callback due to a higher probability of data loss.

Parameters

<i>pEyeImage-Callback-Function</i>	pointer to EyeImageCallbackFunction
------------------------------------	-------------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_SetLicense (const char * *licenseKey*)

Validates the customer license (only for OEM devices). See also [iV_Connect](#), [iV_ConnectLocal](#), [iV_ContinueEyetracking](#), [iV_Disconnect](#), [iV_GetDeviceName](#), [iV_GetFeatureKey](#), [iV_GetLicenseDueDate](#), [iV_GetSerialNumber](#), [iV_GetSpeedModes](#), [iV_GetSystemInfo](#), [iV_IsConnected](#), [iV_PauseEyetracking](#), [iV_Quit](#), [iV_SetConnectionTimeout](#), [iV_SetLicense](#), [iV_SetSpeedMode](#), [iV_Start](#).

Parameters

<i>licenseKey</i>	provided license key
-------------------	----------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_SetLogger (int *logLevel*, char * *filename*)

Defines the logging behavior of iView X SDK.

Parameters

<i>logLevel</i>	log level multiple log levels can can be combined
<i>filename</i>	filename of log file

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_ACCESS_TO_FILE</i>	failed to access log file

int iV_SetREDGeometry (struct REDGeometryStruct * *redGeometry*)

Define the RED and RED-m stand alone and monitor integrated geometry. See chapter [Setting up RED Geometry](#) in the iView X SDK Manual and [iV_DeleteREDGeometry](#), [iV_GetCurrentREDGeometry](#), [iV_GetGeometryProfiles](#), [iV_GetREDGeometry](#), [iV_SelectREDGeometry](#), [iV_SetREDGeometry](#) for details.

Parameters

<i>redGeometry</i>	see reference information for REDGeometryStruct
--------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

[int iV_SetResolution \(int *stimulusWidth*, int *stimulusHeight* \)](#)

Defines a fixed resolution independent to the screen resolution of chosen display device defined in [iV_SetupCalibration](#) function.

Parameters

<i>stimulusWidth</i>	horizontal resolution of stimulus screen [pixel]
<i>stimulusHeight</i>	vertical resolution of stimulus screen [pixel]

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

[int iV_SetSampleCallback \(pDLLSetSample *pSampleCallbackFunction* \)](#)

Sets a callback function for the raw sample data. The function will be called if iView eye tracking server has calculated a new data sample. Important note: Algorithms with high processor usage and long calculation time shouldn't run within this callback due to a higher probability of data loss. See also [iV_GetCurrentTimestamp](#), [iV_GetEvent](#), [iV_GetEvent32](#), [iV_GetSample](#), [iV_GetSample32](#), [iV_GetTrackingStatus](#), [iV_SetEventCallback](#), [iV_SetEventDetectionParameter](#), [iV_SetSampleCallback](#).

Parameters

<i>pSample-Callback-Function</i>	pointer to SampleCallbackFunction
----------------------------------	-----------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_SetSceneVideoCallback (pDLLSetSceneVideo pSceneVideoCallbackFunction)

Sets a callback function for the scene video image data. The function will be called if a new scene video image is available. The image format is RGB 24bpp. Important note: Algorithms with high processor usage and long calculation time shouldn't run within this callback due to a higher probability of data loss.

Parameters

<i>pSceneVideo- Callback- Function</i>	pointer to SceneVideoCallbackFunction
--	---------------------------------------

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_SetSpeedMode (int speedMode)

This function requests the iView eye tracking server to switch the eye tracking frequency to the specified value.

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_SetTrackingMonitorCallback (pDLLSetTrackingMonitor pTrackingMonitorCallbackFunction)

Sets a callback function for the tracking monitor image data. The function will be called if a new tracking monitor image was calculated. The image format is RGB 24bpp. Important note: Algorithms with high processor usage and long calculation time shouldn't run within this callback due to a higher probability of data loss.

Parameters

<i>pTracking-Monitor-Callback-Function</i>	pointer to TrackingMonitorCallbackFunction
--	--

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_SetTrackingParameter (int *ET_PARAM_EYE*, int *ET_PARAM*, int *value*)

Sets iView eye tracking server tracking parameters. See [Eye Tracking Parameter](#) subsection and iView eye tracking server manual for further explanations.

Parameters

<i>ET_PARAM_EYE</i>	select specific eye
<i>ET_PARAM</i>	select parameter that shall be set
<i>value</i>	new value for selected parameter

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_FEATURE_NOT_LICENSED</i>	the features requested by means of the <i>ET_PARAM</i> parameter is not licensed

int iV_SetupCalibration (struct CalibrationStruct * *calibrationData*)

Sets the calibration and validation visualization parameter. See also [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQuality-Image](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibration-Callback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Parameters

<i>calibrationData</i>	see reference information for "CalibrationStruct"
------------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_WRONG_CALIBRATION_METHOD</i>	eye tracking device required for this calibration method is not connected
<i>ERR_FEATURE_NOT_LICENSED</i>	at least one of the features requested by means of the calibrationData parameters are not licensed

int iV_SetupDebugMode (int enableDebugMode)

Enables or disables the debug mode for the current connection. The debug mode disables the automatic connection termination after 5 seconds of an unresponsive server or client. This can happen e.g. during debugging a client application. Beware: the debug mode must not be enabled for production code, as it makes the connection status detection of all API functions unreliable!

Parameters

<i>enableDebugMode</i>	specifies whether the debug mode shall be enabled (1) or disabled (0)
------------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_SetupLptRecording (const char * portName, int enableRecording)

Enables or disables the LPT signal recording functionality. When enabling the LPT port signal recording the LPT port name has to be specified. The LPT ports available for recording can be queried by using the [iV_GetAvailableLptPorts](#) API function. The function must not be called during a running recording.

Parameters

<i>portName</i>	a string referencing the LPT port to be used. Appropriate values can be queried using the iV_GetAvailableLptPorts function
<i>enableRecording</i>	specifies whether recording of LPT port signal shall be enabled (1) or disabled (0)

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	the specified port is not available

<i>ERR_IVIEWX_ACCESS_INCOMPLETE</i>	recording in progress
<i>ERR_NOT_CONNECTED</i>	no connection established

int iV_ShowAccuracyMonitor ()

The validated accuracy results will be visualized in a separate window. Before the image can be drawn the calibration needs to be performed with [iV_Calibrate](#) and validated with [iV_Validate](#). See also [iV_GetAccuracy](#), [iV_GetAccuracyImage](#), [iV_GetGazeChannelQuality](#), [iV_HideAccuracyMonitor](#), [iV_ShowAccuracyMonitor](#), [iV_Validate](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_NO_VALID_DATA</i>	no data available
<i>RET_WINDOW_IS_OPEN</i>	window is already open
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_NOT_CALIBRATED</i>	system is not calibrated
<i>ERR_NOT_VALIDATED</i>	system is not validated

int iV_ShowEyeImageMonitor ()

Visualizes eye image in a separate window while the participant will be tracked.

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_WINDOW_IS_OPEN</i>	window is already open
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

int iV_ShowSceneVideoMonitor ()

Visualizes scene video in separate window (available for HED devices only). See also [iV_GetSceneVideo](#), [iV_HideSceneVideoMonitor](#), [iV_SetSceneVideoCallback](#), [iV_ShowSceneVideoMonitor](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_WINDOW_IS_OPEN</i>	window is already open
<i>ERR_NOT_CONNECTED</i>	no connection established

<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
-------------------------	---

int iV_ShowTrackingMonitor ()

Visualizes RED tracking monitor in a separate window. It shows the position of the participant related to the eye tracking device and indicates (using arrows) if the participant is not positioned in the center of the tracking head box.

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>RET_WINDOW_IS_OPEN</i>	window is already open
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

int iV_Start (enum ETApplication etApplication)

Starts the iView eye tracking server application. Depending on the PC, it may take several seconds to start the iView eye tracking server application. The connection needs to be established separately using [iV_Connect](#). The connection timeout can be extended using [iV_SetConnectionTimeout](#). See also [iV_Connect](#), [iV_ConnectLocal](#), [iV_ContinueEyetracking](#), [iV_Disconnect](#), [iV_GetDeviceName](#), [iV_GetFeatureKey](#), [iV_GetLicenseDueDate](#), [iV_GetSerialNumber](#), [iV_GetSpeedModes](#), [iV_GetSystemInfo](#), [iV_IsConnected](#), [iV_PauseEyetracking](#), [iV_Quit](#), [iV_SetConnectionTimeout](#), [iV_SetLicense](#), [iV_SetSpeedMode](#), [iV_Start](#).

Parameters

<i>etApplication</i>	the iView eye tracking server application which will be started
----------------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_COULD_NOT_CONNECT</i>	failed to establish connection
<i>ERR_IVIEWX_NOT_FOUND</i>	failed to start iViewX application
<i>ERR_CAMERA_NOT_FOUND</i>	no eye tracker connected

int iV_StartRecording ()

Starts gaze data recording and scene video recording (if connected eye tracking device is "HED"). [iV_StartRecording](#) does not return until gaze and scene video recording is started. The data streaming

needs to be stopped by using [iV_StopRecording](#) before it can be saved using [iV_SaveData](#). See also [iV_ClearRecordingBuffer](#), [iV_ContinueRecording](#), [iV_PauseRecording](#), [iV_SaveData](#), [iV_SendImageMessage](#), [iV_StartRecording](#), [iV_StopRecording](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_RECORDING_DATA_BUFFER</i>	recording is activated
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

[int iV_StopRecording \(\)](#)

Stops gaze data recording and scene video recording (if connected eye tracking device is "HED"). [iV_StopRecording](#) does not return until gaze and scene video recording is stopped. This function needs to be called before the data can be saved using [iV_SaveData](#). See also [iV_ClearRecordingBuffer](#), [iV_ContinueRecording](#), [iV_PauseRecording](#), [iV_SaveData](#), [iV_SendImageMessage](#), [iV_StartRecording](#), [iV_StopRecording](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected
<i>ERR_EMPTY_DATA_BUFFER</i>	recording buffer is empty
<i>ERR_FEATURE_NOT_LICENSED</i>	feature not covered by your license

[int iV_TestTTL \(int value \)](#)

Sends a TTL value to defined port. Define a port with [iV_DefineAOIPort](#). See also [iV_ClearAOI](#), [iV_DefineAOI](#), [iV_DefineAOIPort](#), [iV_DisableAOI](#), [iV_DisableAOIGroup](#), [iV_EnableAOI](#), [iV_EnableAOIGroup](#), [iV_GetAOIOutputValue](#), [iV_ReleaseAOIPort](#), [iV_RemoveAOI](#), [iV_SetAOIHitCallback](#), [iV_TestTTL](#).

Parameters

<i>value</i>	value which will be sends out as TTL signal
--------------	---

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_WRONG_PARAMETER</i>	parameter out of range

int iV_Validate ()

Starts a validation procedure. To proceed, the participant needs to be tracked and has to fixate on the validation point. Depending on the validation settings (which can be changed using [iV_SetupCalibration](#) and [iV_SetUseCalibrationKeys](#)) the user can accept the validation points manually (by pressing SPACE or calling [iV_AcceptCalibrationPoint](#)) or abort the validation (by pressing ESC or calling [iV_AbortCalibration](#)). If the validation will be visualized by the API ([CalibrationStruct::visualization](#) is set to 1) the function won't return until the validation has been finished (closed automatically) or aborted (ESC). If the the [CalibrationStruct::visualization](#) is set to 0 [iV_Validate](#) returns immediately. The user has to care about the visualization of validation points. Information about the current validation point can be retrieved with [iV_GetCurrentCalibrationPoint](#) or with setting up the calibration callback using [iV_SetCalibrationCallback](#).

See also [iV_GetAccuracy](#), [iV_GetAccuracyImage](#), [iV_GetGazeChannelQuality](#), [iV_HideAccuracyMonitor](#), [iV_ShowAccuracyMonitor](#), [iV_Validate](#), [iV_AbortCalibration](#), [iV_AbortCalibrationPoint](#), [iV_AcceptCalibrationPoint](#), [iV_Calibrate](#), [iV_ChangeCalibrationPoint](#), [iV_GetCalibrationParameter](#), [iV_GetCalibrationPoint](#), [iV_GetCalibrationQuality](#), [iV_GetCalibrationQualityImage](#), [iV_GetCalibrationStatus](#), [iV_GetCurrentCalibrationPoint](#), [iV_GetUseCalibrationKeys](#), [iV_LoadCalibration](#), [iV_RecalibrateOnePoint](#), [iV_ResetCalibrationPoints](#), [iV_SaveCalibration](#), [iV_SetCalibrationCallback](#), [iV_SetResolution](#), [iV_SetupCalibration](#), [iV_SetUseCalibrationKeys](#).

Return values

<i>RET_SUCCESS</i>	intended functionality has been fulfilled
<i>ERR_NOT_CONNECTED</i>	no connection established
<i>ERR_NOT_CALIBRATED</i>	system is not calibrated
<i>ERR_WRONG_DEVICE</i>	eye tracking device required for this function is not connected

2.5 Functions Grouped by Topic

Topic	List of Related Functions
AOI Trigger	iV_ClearAOI , iV_DefineAOI , iV_DefineAOIPort , iV_DisableAOI , iV_DisableAOIGroup , iV_EnableAOI , iV_EnableAOIGroup , iV_GetAOIOutputValue , iV_ReleaseAOIPort , iV_RemoveAOI , iV_SetAOIHitCallback , iV_TestTTL
Calibration	iV_AbortCalibration , iV_AbortCalibrationPoint , iV_AcceptCalibrationPoint , iV_Calibrate , iV_ChangeCalibrationPoint , iV_GetCalibrationParameter , iV_GetCalibrationPoint , iV_GetCalibrationQuality , iV_GetCalibrationQualityImage , iV_GetCalibrationStatus , iV_GetCurrentCalibrationPoint , iV_GetUseCalibrationKeys , iV_LoadCalibration , iV_RecalibrateOnePoint , iV_ResetCalibrationPoints , iV_SaveCalibration , iV_SetCalibrationCallback , iV_SetResolution , iV_SetupCalibration , iV_SetUseCalibrationKeys
Data Acquisition	iV_GetCurrentTimestamp , iV_GetEvent , iV_GetEvent32 , iV_GetSample , iV_GetSample32 , iV_GetTrackingStatus , iV_SetEventCallback , iV_SetEventDetectionParameter , iV_SetSampleCallback
Eye Data Recording	iV_ClearRecordingBuffer , iV_ContinueRecording , iV_PauseRecording , iV_SaveData , iV_SendImageMessage , iV_StartRecording , iV_StopRecording
Eye Image Handling	iV_GetEyeImage , iV_HideEyeImageMonitor , iV_SetEyeImageCallback , iV_ShowEyeImageMonitor
Gaze Data Filter	iV_DisableGazeDataFilter , iV_EnableGazeDataFilter , iV_ConfigureFilter
Geometry RED	iV_DeleteREDGeometry , iV_GetCurrentREDGeometry , iV_GetGeometryProfiles , iV_GetREDGeometry , iV_SelectREDGeometry , iV_SetREDGeometry
Logging	iV_Log , iV_SetLogger

Other	iV_SendCommand, iV_SetTrackingParameter
RED Tracking Monitor Handling	iV_GetTrackingMonitor, iV_HideTrackingMonitor, iV_SetTrackingMonitorCallback, iV_ShowTrackingMonitor
System Start and Stop, System Information and Connection	iV_Connect, iV_ConnectLocal, iV_ContinueEyetracking, iV_Disconnect, iV_GetDeviceName, iV_GetFeatureKey, iV_GetLicenseDueDate, iV_GetSerialNumber, iV_GetSpeedModes, iV_GetSystemInfo, iV_IsConnected, iV_PauseEyetracking, iV_Quit, iV_SetConnectionTimeout, iV_SetLicense, iV_SetSpeedMode, iV_Start
Validation	iV_GetAccuracy, iV_GetAccuracyImage, iV_GetGazeChannelQuality, iV_HideAccuracyMonitor, iV_ShowAccuracyMonitor, iV_Validate

2.6 Function Return Values

The Return values listed in the header defines all possible return codes which can be returned by the API functions.

Return Code	Value	Description
RET_SUCCESS	1	intended functionality has been fulfilled
RET_NO_VALID_DATA	2	no new data available
RET_CALIBRATION_ABORTED	3	calibration / validation was aborted during progress
RET_SERVER_IS_RUNNING	4	server is running
RET_CALIBRATION_NOT_IN_PROGRESS	5	calibration is not in progress
RET_WINDOW_IS_OPEN	11	window is open
RET_WINDOW_IS_CLOSED	12	window is closed
ERR_COULD_NOT_CONNECT	100	failed to establish connection
ERR_NOT_CONNECTED	101	no connection established
ERR_NOT_CALIBRATED	102	system is not calibrated
ERR_NOT_VALIDATED	103	system is not validated
ERR_EYETRACKING_APPLICATION_NOT_RUNNING	104	no eye tracking application running
ERR_WRONG_COMMUNICATION_PARAMETER	105	failed to establish connection
ERR_WRONG_DEVICE	111	eye tracking device required for this function is not connected
ERR_WRONG_PARAMETER	112	parameter out of range
ERR_WRONG_CALIBRATION_METHOD	113	eye tracking device required for this calibration method is not connected
ERR_CALIBRATION_TIMEOUT	114	calibration timeout occurred
ERR_TRACKING_NOT_STABLE	115	eye tracking is not stable
ERR_INSUFFICIENT_BUFFER_SIZE	116	insufficient buffer size
ERR_CREATE_SOCKET	121	cannot create socket
ERR_CONNECT_SOCKET	122	cannot connect with socket
ERR_BIND_SOCKET	123	the defined port is blocked

ERR_DELETE_SOCKET	124	failed to delete sockets
ERR_NO_RESPONSE_FROM_IVIEWX	131	iView X (eyetracking-server) application was not able to response to current request
ERR_INVALID_IVIEWX_VERSION	132	invalid version of iView X (eyetracking-server)
ERR_WRONG_IVIEWX_VERSION	133	wrong version of iView X (eyetracking-server) application
ERR_ACCESS_TO_FILE	171	failed to access log file
ERR_SOCKET_CONNECTION	181	socket connection failed
ERR_EMPTY_DATA_BUFFER	191	recording buffer is empty
ERR_RECORDING_DATA_BUFFER	192	recording is activated
ERR_FULL_DATA_BUFFER	193	data buffer is full
ERR_IVIEWX_IS_NOT_READY	194	iView X (eyetracking-server) application is not ready to record buffer
ERR_PAUSED_DATA_BUFFER	195	paused data buffer
ERR_IVIEWX_NOT_FOUND	201	iView X (eyetracking-server) application was not found
ERR_IVIEWX_PATH_NOT_FOUND	202	path for file does not exist
ERR_IVIEWX_ACCESS_DENIED	203	access denied
ERR_IVIEWX_ACCESS_INCOMPLETE	204	access incomplete
ERR_IVIEWX_OUT_OF_MEMORY	205	out of memory
ERR_CAMERA_NOT_FOUND	211	failed to access eye tracking device
ERR_WRONG_CAMERA	212	failed to access eye tracking device
ERR_WRONG_CAMERA_PORT	213	failed to access port connected to eye tracking device
ERR_COULD_NOT_OPEN_PORT	220	failed to open port
ERR_COULD_NOT_CLOSE_PORT	221	failed to close port
ERR_AOI_ACCESS	222	failed to access AOI data

ERR_AOI_NOT_DEFINED	223	AOI not defined
ERR_FEATURE_NOT_LICENSED	250	failed to access requested feature
ERR_DEPRECATED_FUNCTION	300	function is deprecated
ERR_INITIALIZATION	400	function or dll not initialized

2.7 Eye Tracking Parameter

With ET_PARAM_ and function [iV_SetTrackingParameter](#) it is possible to change iView X and eyetracking-server tracking parameters, for example pupil threshold and corneal reflex thresholds, eye image contours, and other parameters. Important note: This function can strongly affect tracking stability of your iView X and eyetracking-server system. Only experienced users should use this function.

Parameter	Value	Description
ET_PARAM_EYE_LEFT	0	set parameter for the left eye
ET_PARAM_EYE_RIGHT	1	set parameter for the left eye
ET_PARAM_EYE_BOTH	2	set parameter for both eyes
ET_PARAM_PUPIL_THRESHOLD	0	set pupil threshold parameter
ET_PARAM_REFLEX_THRESHOLD	1	set reflex threshold parameter
ET_PARAM_SHOW_AOI	2	enabling/disabling AOI overlays
ET_PARAM_SHOW_CONTOUR	3	enabling/disabling eye contour overlays
ET_PARAM_SHOW_PUPIL	4	enabling/disabling pupil center overlays
ET_PARAM_SHOW_REFLEX	5	enabling/disabling reflex center overlays
ET_PARAM_DYNAMIC_THRESHOLD	6	enabling/disabling dynamic pupil threshold
ET_PARAM_PUPIL_AREA	11	set pupil area parameter
ET_PARAM_PUPIL_PERIMETER	12	set pupil perimeter
ET_PARAM_PUPIL_DENSITY	13	set pupil density parameter
ET_PARAM_REFLEX_PERIMETER	14	set reflex perimeter
ET_PARAM_REFLEX_PUPIL_DISTANCE	15	set reflex pupil distance parameter
ET_PARAM_MONOCULAR	16	set tracking mode to monocular
ET_PARAM_SMARTBINOCULAR	17	set tracking mode to smart binocular
ET_PARAM_BINOCULAR	18	set tracking mode to binocular
ET_PARAM_SMARTTRACKING	19	set tracking mode to smart tracking

2.8 Functions implemented in EyeTracker2Impl for NBS Presentation

Function	Supported
accept_point	-
buffer_position	X
calibration	X
validation (EyeTracker2CalibrationType = 2)	X
clear_buffer	X
event_count	X
get_aoi_event	-
get_blink_event	-
get_calibration_point	-
get_fixation_event	X
get_parameter	-
get_position_data	X
get_pupil_data	X
get_saccade_event	-
get_status	-
get_trigger	-
is_recording	-
last_aoi_event	-
last_blink_event	-
last_fixation_event	X
last_position_data	X
last_pupil_data	X
last_saccade_event	-
new_aoi_events	-
new_blink_events	-
new_fixation_events	X
new_position_data	X
new_pupil_data	X
new_saccade_events	-
send_command	-
send_message	X
send_string	-
send_trigger	-

set_abort_on_error	X
set_aoi_set	-
set_default_data_set	X
set_max_buffer_size	X
set_parameter	-
set_recording	X
start_calibration	-
start_tracking	X
start_data	X
stop_calibration	-
stop_tracking	X
stop_data	X
supports	X
trigger_count	-
version	X

2.9 Function and Device Overview

The table below provides an overview of the various functions available in the iView X™ SDK along with their corresponding supported SMI eye tracking devices. More detailed information pertaining to these functions follows in the iView X™ SDK Reference section. A device-specific description on supported modi for the functions can be found in the respective device manual

Function	RE-D250mo	REDn Professional	REDn Scientific	RED	RED-m	RED-mx	Hi-Speed/-Primate	HED	MRI/MEG
iV_-Abort-Calibration	X	X	X	X	X	X	X	-	X
iV_-Abort-Calibration-Point	X	X	X	-	-	-	-	-	-
iV_-Accept-Calibration-Point	X	X	X	X	X	X	X	-	X
iV_-Calibrate	X	X	X	X	X	X	X	-	X
iV_-Change-Calibration-Point	X	X	X	X	X	X	X	X	X
iV_-Clear-AOI	X	X	X	X	X	X	X	-	X
iV_-Clear-Recording-Buffer	X	X	X	X	X	X	X	X	X
iV_-Configure-Filter	X	-	X	X	X	X	X	X	X
iV_-Connect	X	X	X	X	X	X	X	X	X
iV_-Connect-Local	X	X	X	-	X	X	-	-	-

iV_- Continue- Eyetracking	X	X	X	-	X	X	-	-	-
iV_- Continue- Recording	X	X	X	X	X	X	X	X	X
iV_- Define- AOI	-	-	-	X	X	X	X	-	X
iV_- Define- AOI- Port	-	-	-	X	X	X	X	-	X
iV_- Delete- RED- Geometry	X	X	X	X	X	X	-	-	-
iV_- Disable- AOI	-	-	-	X	X	X	X	-	X
iV_- Disable- AOI- Group	-	-	-	X	X	X	X	-	X
iV_- Disable- Gaze- Data- Filter	X	X	X	X	X	X	X	-	X
iV_- Disable- Processor- High- Performance- Mode	X	X	X	-	X	X	-	-	-
iV_- Disconnect	X	X	X	X	X	X	X	X	X
iV_- Enable- AOI	-	-	-	X	X	X	X	-	X

iV_- Enable- AOI- Group	-	-	-	X	X	X	X	-	X
iV_- Enable- Gaze- Data- Filter	X	X	X	X	X	X	X	-	X
iV_- Enable- Processor- High- Performance- Mode	X	X	X	-	X	X	-	-	-
iV_- Get- Accuracy	X	X	X	X	X	X	X	-	X
iV_- Get- Accuracy- Image	X	X	X	X	X	X	X	-	X
iV_- GetA- OI- Output- Value	X	X	X	X	X	X	X	-	X
iV_- Get- Calibration- Parameter	X	X	X	-	-	-	-	-	-
iV_- Get- Calibration- Point	X	X	X	X	X	X	X	X	X
iV_- Get- Calibration- Quality	X	X	X	-	-	-	-	-	-

iV_- Get- Calibration- Quality- Image	X	X	X	-	-	-	-	-	-
iV_- Get- Calibration- Status	X	X	X	X	X	X	X	X	X
iV_- Get- Current- Calibration- Point	X	X	X	X	X	X	X	X	X
iV_- Get- Current- RED- Geometry	X	X	X	X	X	X	-	-	-
iV_- Get- Current- Timestamp	X	X	X	X	X	X	X	X	X
iV_- Get- Device- Name	X	X	X	-	X	X	-	-	-
iV_- Get- Event	X	-	X	X	X	X	X	-	X
iV_- Get- Event32	X	-	X	X	X	X	X	-	X
iV_- Get- Eye- Image	X	-	X	X	X	-	X	X	X
iV_- Get- Feature- Key	X	X	X	-	-	-	-	-	-

iV_- Get- Gaze- Channel- Quality	X	X	X	-	-	-	-	-	-
iV_- Get- Geometry- Profiles	X	X	X	-	-	-	-	-	-
iV_- Get- License- Due- Date	X	X	X	X	X	X	X	X	X
iV_- GetR- ED- Geometry	X	X	X	X	X	X	-	-	-
iV_- Get- Sample	X	X	X	X	X	X	X	X	X
iV_- Get- Sample32	X	X	X	X	X	X	X	X	X
iV_- Get- Scene- Video	-	-	-	-	-	-	-	X	-
iV_- Get- Serial- Number	X	X	X	-	X	X	-	-	-
iV_- Get- Speed- Modes	X	X	X	-	-	-	-	-	-
iV_- Get- System- Info	X	X	X	X	X	X	X	X	X

iV_- Get- Tracking- Monitor	X	X	X	X	X	X	-	-	-
iV_- Get- Tracking- Status	X	X	X	X	X	X	X	X	X
iV_- Get- Use- Calibration- Keys	X	X	X	-	-	-	-	-	-
iV_- Hide- Accuracy- Monitor	X	X	X	X	X	X	X	X	X
iV_- Hide- Eye- Image- Monitor	X	-	X	X	X	X	X	X	X
iV_- Hide- Scene- Video- Monitor	-	-	-	-	-	-	-	X	-
iV_- Hide- Tracking- Monitor	X	X	X	X	X	X	-	-	-
iV_Is- Connected	X	X	X	X	X	X	X	X	X
iV_- Load- Calibration	X	X	X	X	X	X	X	-	X
iV_Log	X	X	X	X	X	X	X	X	X
iV_- Pause- Eyetracking	X	X	X	-	X	X	-	-	-

iV_- Pause- Recording	X	X	X	X	X	X	X	X	X
iV_Quit	X	X	X	X	X	X	X	X	X
iV_- Recalibrate- One- Point	X	X	X	-	-	-	-	-	-
iV_- Release- AOI- Port	-	-	-	X	X	X	X	-	X
iV_- Remove- AOI	-	-	-	X	X	X	X	-	X
iV_- Reset- Calibration- Points	X	X	X	X	X	X	X	X	X
iV_- Save- Calibration	X	X	X	X	X	X	X	-	X
iV_- Save- Data	X	X	X	X	X	X	X	X	X
iV_- Select- RED- Geometry	X	X	X	X	X	X	-	-	-
iV_- Send- Command	X	X	X	X	X	X	X	X	X
iV_- Send- Image- Message	X	X	X	X	X	X	X	X	X
iV_Set- AOIHit- Callback	-	-	-	X	X	X	X	-	X

iV_Set- Calibration- Callback	X	X	X	X	X	X	X	-	X
iV_Set- Connection- Timeout	X	-	X	X	X	X	X	X	X
iV_Set- Event- Callback	X	-	X	X	X	X	X	-	X
iV_Set- Event- Detection- Parameter	X	-	X	X	X	X	X	-	X
iV_Set- Eye- Image- Callback	X	-	X	X	X	-	X	X	X
iV_Set- License	X	-	X	-	-	-	-	-	-
iV_Set- Logger	X	X	X	X	X	X	X	X	X
iV_Set- RED- Geometry	X	X	X	X	X	X	-	-	-
iV_Set- Resolution	X	X	X	X	X	X	X	-	X
iV_Set- Sample- Callback	X	X	X	X	X	X	X	X	X
iV_Set- Scene- Video- Callback	-	-	-	-	-	-	-	X	-
iV_Set- Speed- Mode	X	X	X	-	-	-	-	-	-
iV_Set- Tracking- Monitor- Callback	X	X	X	X	X	X	-	-	-

iV_Set-Tracking-Parameter	X	X	X	-	X	X	X	X	X
iV_-Setup-Calibration	X	X	X	X	X	X	X	-	X
iV_Set-Use-Calibration-Keys	X	X	X	-	-	-	-	-	-
iV_-Show-Accuracy-Monitor	X	X	X	X	X	X	X	-	X
iV_-Show-Eye-Image-Monitor	X	-	X	X	X	-	X	X	X
iV_-Show-Scene-Video-Monitor	-	-	-	-	-	-	-	X	-
iV_-Show-Tracking-Monitor	X	X	X	X	X	X	-	-	-
iV_-Start	X	X	X	X	X	X	X	X	X
iV_-Start-Recording	X	X	X	X	X	X	X	X	X
iV_-Stop-Recording	X	X	X	X	X	X	X	X	X
iV_-Test-TL	-	-	-	X	X	X	X	-	X

iV_- Validate	X	X	X	X	X	X	X	-	X
------------------	---	---	---	---	---	---	---	---	---

Chapter 3

Appendix

3.1 LICENSE AGREEMENT FOR THE SMI SOFTWARE DEVELOPMENT KITS (“-SDK”) PROVIDED FREE OF CHARGE

IMPORTANT – PLEASE READ CAREFULLY: This license agreement (“Agreement”) is an agreement between you (either an individual or Your Company, “Licensee”) and SensoMotoric Instruments Gesellschaft für innovative Sensorik mbH, Warthestraße 21, 14513 Teltow, Germany (“SMI”). The “-Licensed Materials” provided to Licensee free of charge subject to this Agreement include the SDK and any “on-line”, electronic or written documentation associated with the SDK, or any portion thereof (the “Documentation”), as well as any updates or upgrades to the SDK and Documentation, if any, or any portion thereof, provided to Licensee at SMI’s sole discretion. The application of conflicting general terms and conditions of Licensee shall be excluded. This applies irrespective of whether or not such terms and conditions have been expressly rejected by SMI or whether SMI, having knowledge of such conflicting terms and conditions, has accepted or effects contractual performance without reservation. By clicking “Accept”, you agree to abide by the following provisions. This Agreement is displayed for you to read and accept prior to using the Licensed Materials. If you do not agree with these provisions, do not install, download or use the Licensed Materials.

1) License. Subject to the terms of this Agreement, SMI hereby grants and Licensee accepts a non-transferable and non-exclusive license without the right to sublicense for the use of the Licensed Materials only for (i) Licensee’s operations, (ii) the development of applications for SMI Eye Tracking Devices, (iii) in accordance with the Documentation and (iv) only by one (1) concurrent user. Licensee may make one (1) copy of the SDK in machine readable form for backup purposes only; every notice on the original will be replicated on the copy. Installation of the SDK is Licensee’s sole responsibility. The Licensed Materials may be protected by technical means as explained in the user manual, if any. Licensee is not entitled to rent, lease or otherwise make available the SDK to third parties on a non-permanent commercial basis (including as part of any software as a service or application service provider offering), except with the prior written consent of SMI.

2) Rights in Licensed Materials. With the exception of the limited license rights granted to Licensee under this Agreement, title to and ownership in the Licensed Materials and all proprietary rights with respect to the Licensed Materials remain exclusively with SMI. Title to and ownership in Licensee’s

application software that makes calls to the SDK but does not contain the SDK or any portion thereof remain with Licensee, but such application software may not be licensed or otherwise transferred to third parties without SMI's prior written consent. Confidentiality. Licensed Materials are proprietary to SMI and constitute SMI trade and business secrets. 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. 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 (i) is or becomes generally known or available to the public without any breach of the confidentiality obligation by Licensee, (ii) was already known to Licensee prior to the disclosure by SMI, or (iii) was rightfully acquired by Licensee from a third party without a breach of a confidentiality obligation towards SMI. In case of a dispute, Licensee has the burden of proof that the Licensed Materials and/or any portion thereof fall under one of these exceptions. Should 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, Licensee shall, to the extent permitted under applicable law, inform SMI without undue delay and undertake all possible measures to safeguard secrecy.

3) No Reverse Engineering. Licensee shall not, and shall not allow any third party to, decompile, disassemble or otherwise reverse engineer or by any means whatsoever attempt to reconstruct or discover any source code or underlying ideas, algorithms, file formats or programming or interoperability interfaces of the SDK or of any files contained or generated using the SDK.

4) Warranty. The Licensed Materials are provided free of any charge and "as is" without any warranty of any kind.

5) Liability Limitations.

a) SMI shall only be liable for damages caused by willful intent or gross negligence.

b) With the exception of claims for damages in the event of (i) an injury to life, body or health, (ii) liability under the Product Liability Act [Produkthaftungsgesetz], and (iii) fraudulently concealed defects, the above limitation of liability shall apply to all claims for damages, irrespective of their legal basis, including but not limited to all claims based on breach of contract or tort.

c) The above limitation of liability also applies in case of Licensee's claims for damages against SMI's employees, statutory representatives [gesetzliche Vertreter] or agents [Erfüllungsgehilfen].

6) Licensee Indemnity. Licensee will defend and indemnify SMI, and hold it harmless from all costs, including attorney's fees, arising from any claim that may be made against SMI by any third party as a result of Licensee's use of Licensed Materials.

7) Export Restriction. Licensee will not remove or export from Germany or from the country Licensed Materials were originally transmitted and/or shipped to by SMI or re-export from anywhere any part of the Licensed Materials or any direct product of the SDK except in compliance with all applicable export laws and regulations, including without limitation, those of the U.S. Department of Commerce.

8) Non-Waiver; Severability; Non-Assignment. The delay or failure of either party to exercise any right provided in this Agreement shall not be deemed a waiver. If any provision of this Agreement is held invalid, all others shall remain in force. Except as expressly set forth in this Agreement, Licensee may

not, in whole or in part, assign or otherwise transfer this Agreement or any of its rights or obligations hereunder.

9) Entire Agreement; Written Form Requirement. Unless otherwise specified herein, SMI's General Terms and Conditions for the Supply of Products and Services available at <http://www.smivision.com/en/gaze-and-eye-tracking-systems/support/documents-download.-html> or attached hereto apply. Any supplementary agreements or modifications hereto must be made in writing. This also applies to any waiver of this requirement of written form.

10) Termination. This Agreement may be terminated by written notice (i) by Licensee without cause on 30 (thirty) days written notice; (ii) by SMI, in addition to other remedies SMI may have, if Licensee fails to cure any breach of this Agreement within 30 (thirty) days of written notice thereof; (iii) by SMI if there is a transfer of twenty-five percent (25%) or more of the ownership interest in Licensee, which in good faith is not acceptable to SMI, and by either party if the other party ceases to do business in the normal course, becomes insolvent, or becomes subject to any bankruptcy, insolvency, or equivalent proceedings. Upon termination by either party for any reason, Licensee shall at SMI's instructions immediately destroy or return to SMI the Licensed Materials and all copies thereof and delete the SDK and all copies thereof from any computer on which the SDK had been installed.

11) Notices. All notices under the Agreement must be in writing and shall be delivered by hand or by overnight courier.

12) Applicable Law and Jurisdiction. German law applies with the exception of its conflict of laws rules. The application of the United Nations Convention on Contracts for the International Sale of Goods (CIS-G) is expressly excluded. The courts of Berlin, Germany, shall have exclusive jurisdiction for any action brought under or in connection with this Agreement.

© Teltow, Germany, 2004-2015 SensoMotoric Instruments GmbH

3.2 Technical Support

Due to the complex nature of SDK's in general and the wide variety of applications that may be created using the iView X™ SDK, it is not always possible to provide in-depth support. However, if you feel there is an error or omission in the iView X™ SDK, please fill out a support request on the SMI website (<http://www.smivision.com/en/gaze-and-eye-tracking-systems/support/support-request.-html>) and we will research the issue. Please note that if you should require technical assistance relating to the SDK and your application, SMI may request or require a copy of your application and elements of your source code. If you are new to programming, we would highly recommend that you consult a general programming guide for your desired language before attempting to use the iView X™ SDK to write your own eyetracking application. The provided examples are included to help you in getting started with developing your software application, but they are not a substitute for programming knowledge.

3.3 About SMI

SensoMotoric Instruments (SMI) is a world leader in dedicated computer vision applications, developing and marketing eye & gaze tracking systems and OEM solutions for a wide range of applications.

Founded in 1991 as a spin-off from academic research, SMI was the first company to offer a commercial, vision-based 3D eye tracking solution. We now have 20 years of experience in developing application-specific solutions in close collaboration with our clients. We serve our customers around the globe from our offices in Teltow, near Berlin, Germany and Boston, USA, backed by a network of trusted local partners in many countries. Our products combine a maximum of performance and usability with the highest possible quality, resulting in high-value solutions for our customers. Our major fields of expertise are: • Eye & gaze tracking systems in research and industry • High speed image processing, and • Eye tracking and registration solutions in ophthalmology. More than 4,000 of our systems installed worldwide are testimony to our continuing success in providing innovative products and outstanding services to the market. While SMI has won several awards, the largest reward for us each year is our trusted business relationships with academia and industry.

Please contact us:

Europe, Asia, Africa, South America, Australia
SensoMotoric Instruments GmbH (SMI)
Warthestraße 21
D-14513 Teltow
Germany
Phone: +49 3328 3955 0
Fax: +49 3328 3955 99
Email: info@smi.de

North American Headquarters
SensoMotoric Instruments, Inc.
28 Atlantic Avenue
236 Lewis Wharf
Boston, MA 02110
USA
Phone: +1 - 617 - 557 - 0010
Fax: +1 - 617 - 507 - 83 19
Toll-Free: 888 SMI USA1
Email: info@smivision.com

Please also visit our home page: <http://www.smivision.com>

Copyright © 2014 SensoMotoric Instruments GmbH

Last updated: December 2014

Index

- AOIRectangleStruct, [46](#)
- AOIStruct, [46](#)
- AccuracyStruct, [45](#)
- Average
 - Enumerations, [43](#)
- calibrationInProgress
 - Enumerations, [42](#)
- calibrationInvalid
 - Enumerations, [42](#)
- calibrationPointIgnored
 - Enumerations, [42](#)
- CalibrationPointQualityStruct, [46](#)
- CalibrationPointStruct, [47](#)
- calibrationPointUnused
 - Enumerations, [41](#)
- calibrationPointUnusedBecauseOfBadQuality
 - Enumerations, [42](#)
- calibrationPointUnusedBecauseOfTimeout
 - Enumerations, [41](#)
- calibrationPointUsed
 - Enumerations, [41](#)
- CalibrationStruct, [47](#)
- calibrationUnknown
 - Enumerations, [42](#)
- calibrationValid
 - Enumerations, [42](#)
- CalibrationPointUsageStatusEnum
 - Enumerations, [41](#)
- CalibrationStatusEnum
 - Enumerations, [42](#)
- Callback Function Types, [54](#)
- Custom
 - Enumerations, [43](#)
- Data Structures, [44](#)
- DateStruct, [48](#)
- ETApplication
 - Enumerations, [42](#)
- ETDevice
 - Enumerations, [42](#)
- Enumerations, [41](#)
 - Average, [43](#)
 - calibrationInProgress, [42](#)
 - calibrationInvalid, [42](#)
 - calibrationPointIgnored, [42](#)
 - calibrationPointUnused, [41](#)
 - calibrationPointUnusedBecauseOfBadQuality, [42](#)
 - calibrationPointUnusedBecauseOfTimeout, [41](#)
 - calibrationPointUsed, [41](#)
 - calibrationUnknown, [42](#)
 - calibrationValid, [42](#)
 - CalibrationPointUsageStatusEnum, [41](#)
 - CalibrationStatusEnum, [42](#)
 - Custom, [43](#)
 - ETApplication, [42](#)
 - ETDevice, [42](#)
 - FilterAction, [43](#)
 - FilterType, [43](#)
 - HED, [43](#)
 - HiSpeed, [42](#)
 - iViewNG, [42](#)
 - iViewX, [42](#)
 - iViewXOEM, [42](#)
 - MRI, [42](#)
 - monitorIntegrated, [43](#)
 - NONE, [42](#)
 - Query, [43](#)
 - RED, [42](#)
 - REDm, [42](#)
 - REDn, [43](#)
 - REDGeometryEnum, [43](#)
 - Set, [43](#)
 - standalone, [43](#)

- EventStruct, [48](#)
- EventStruct32, [49](#)
- Eye Tracking Parameter, [102](#)
- EyeDataStruct, [49](#)
- EyePositionStruct, [49](#)
- FilterAction
 - Enumerations, [43](#)
- FilterType
 - Enumerations, [43](#)
- Function and Device Overview, [105](#)
- Function Return Values, [99](#)
- Functions, [55](#)
 - iV_AbortCalibration, [57](#)
 - iV_AbortCalibrationPoint, [58](#)
 - iV_AcceptCalibrationPoint, [58](#)
 - iV_Calibrate, [58](#)
 - iV_ChangeCalibrationPoint, [59](#)
 - iV_ClearAOI, [59](#)
 - iV_ClearRecordingBuffer, [60](#)
 - iV_ConfigureFilter, [60](#)
 - iV_Connect, [61](#)
 - iV_ConnectLocal, [61](#)
 - iV_ContinueEyetracking, [62](#)
 - iV_ContinueRecording, [62](#)
 - iV_DefineAOI, [62](#)
 - iV_DefineAOIPort, [63](#)
 - iV_DeleteREDGeometry, [63](#)
 - iV_DisableAOI, [63](#)
 - iV_DisableAOIGroup, [64](#)
 - iV_DisableGazeDataFilter, [64](#)
 - iV_DisableProcessorHighPerformanceMode, [64](#)
 - iV_Disconnect, [64](#)
 - iV_EnableAOI, [65](#)
 - iV_EnableAOIGroup, [65](#)
 - iV_EnableGazeDataFilter, [66](#)
 - iV_EnableProcessorHighPerformanceMode, [66](#)
 - iV_GetAOIOutputValue, [67](#)
 - iV_GetAccuracy, [66](#)
 - iV_GetAccuracyImage, [66](#)
 - iV_GetAvailableLptPorts, [67](#)
 - iV_GetCalibrationParameter, [68](#)
 - iV_GetCalibrationPoint, [68](#)
 - iV_GetCalibrationQuality, [69](#)
 - iV_GetCalibrationQualityImage, [69](#)
 - iV_GetCalibrationStatus, [69](#)
 - iV_GetCurrentCalibrationPoint, [70](#)
 - iV_GetCurrentREDGeometry, [70](#)
 - iV_GetCurrentTimestamp, [71](#)
 - iV_GetDeviceName, [71](#)
 - iV_GetEvent, [71](#)
 - iV_GetEvent32, [72](#)
 - iV_GetEyeImage, [72](#)
 - iV_GetFeatureKey, [73](#)
 - iV_GetGazeChannelQuality, [73](#)
 - iV_GetGeometryProfiles, [73](#)
 - iV_GetLicenseDueDate, [74](#)
 - iV_GetREDGeometry, [74](#)
 - iV_GetSample, [74](#)
 - iV_GetSample32, [75](#)
 - iV_GetSceneVideo, [75](#)
 - iV_GetSerialNumber, [76](#)
 - iV_GetSpeedModes, [76](#)
 - iV_GetSystemInfo, [76](#)
 - iV_GetTrackingMonitor, [77](#)
 - iV_GetTrackingStatus, [77](#)
 - iV_HideAccuracyMonitor, [78](#)
 - iV_HideEyeImageMonitor, [78](#)
 - iV_HideSceneVideoMonitor, [78](#)
 - iV_HideTrackingMonitor, [78](#)
 - iV_IsConnected, [79](#)
 - iV_LoadCalibration, [79](#)
 - iV_Log, [79](#)
 - iV_PauseEyetracking, [80](#)
 - iV_PauseRecording, [80](#)
 - iV_Quit, [80](#)
 - iV_RecalibrateOnePoint, [81](#)
 - iV_ReleaseAOIPort, [82](#)
 - iV_RemoveAOI, [82](#)
 - iV_ResetCalibrationPoints, [82](#)
 - iV_SaveCalibration, [83](#)
 - iV_SaveData, [83](#)
 - iV_SelectREDGeometry, [84](#)
 - iV_SendCommand, [84](#)
 - iV_SendImageMessage, [84](#)
 - iV_SetAOIHitCallback, [85](#)
 - iV_SetCalibrationCallback, [85](#)
 - iV_SetConnectionTimeout, [86](#)

- iV_SetEventCallback, [86](#)
- iV_SetEventDetectionParameter, [87](#)
- iV_SetEyeImageCallback, [87](#)
- iV_SetLicense, [88](#)
- iV_SetLogger, [88](#)
- iV_SetREDGeometry, [88](#)
- iV_SetResolution, [89](#)
- iV_SetSampleCallback, [89](#)
- iV_SetSceneVideoCallback, [90](#)
- iV_SetSpeedMode, [90](#)
- iV_SetTrackingMonitorCallback, [90](#)
- iV_SetTrackingParameter, [91](#)
- iV_SetupCalibration, [91](#)
- iV_SetupDebugMode, [92](#)
- iV_SetupLptRecording, [92](#)
- iV_ShowAccuracyMonitor, [93](#)
- iV_ShowEyeImageMonitor, [93](#)
- iV_ShowSceneVideoMonitor, [93](#)
- iV_ShowTrackingMonitor, [94](#)
- iV_Start, [94](#)
- iV_StartRecording, [94](#)
- iV_StopRecording, [95](#)
- iV_TestTTL, [95](#)
- iV_Validate, [96](#)
- Functions Grouped by Topic, [97](#)
- Functions implemented in EyeTracker2Impl for NB-
S Presentation, [103](#)
- GazeChannelQualityStruct, [50](#)
- HED
 - Enumerations, [43](#)
- HiSpeed
 - Enumerations, [42](#)
- iViewNG
 - Enumerations, [42](#)
- iViewX
 - Enumerations, [42](#)
- iViewXOEM
 - Enumerations, [42](#)
- iV_AbortCalibration
 - Functions, [57](#)
- iV_AbortCalibrationPoint
 - Functions, [58](#)
- iV_AcceptCalibrationPoint
 - Functions, [58](#)
- iV_Calibrate
 - Functions, [58](#)
- iV_ChangeCalibrationPoint
 - Functions, [59](#)
- iV_ClearAOI
 - Functions, [59](#)
- iV_ClearRecordingBuffer
 - Functions, [60](#)
- iV_ConfigureFilter
 - Functions, [60](#)
- iV_Connect
 - Functions, [61](#)
- iV_ConnectLocal
 - Functions, [61](#)
- iV_ContinueEyetracking
 - Functions, [62](#)
- iV_ContinueRecording
 - Functions, [62](#)
- iV_DefineAOI
 - Functions, [62](#)
- iV_DefineAOIPort
 - Functions, [63](#)
- iV_DeleteREDGeometry
 - Functions, [63](#)
- iV_DisableAOI
 - Functions, [63](#)
- iV_DisableAOIGroup
 - Functions, [64](#)
- iV_DisableGazeDataFilter
 - Functions, [64](#)
- iV_DisableProcessorHighPerformanceMode
 - Functions, [64](#)
- iV_Disconnect
 - Functions, [64](#)
- iV_EnableAOI
 - Functions, [65](#)
- iV_EnableAOIGroup
 - Functions, [65](#)
- iV_EnableGazeDataFilter
 - Functions, [66](#)
- iV_EnableProcessorHighPerformanceMode
 - Functions, [66](#)
- iV_GetAOIOutputValue
 - Functions, [67](#)

iV_GetAccuracy	Functions, 66
iV_GetAccuracyImage	Functions, 66
iV_GetAvailableLptPorts	Functions, 67
iV_GetCalibrationParameter	Functions, 68
iV_GetCalibrationPoint	Functions, 68
iV_GetCalibrationQuality	Functions, 69
iV_GetCalibrationQualityImage	Functions, 69
iV_GetCalibrationStatus	Functions, 69
iV_GetCurrentCalibrationPoint	Functions, 70
iV_GetCurrentREDGeometry	Functions, 70
iV_GetCurrentTimestamp	Functions, 71
iV_GetDeviceName	Functions, 71
iV_GetEvent	Functions, 71
iV_GetEvent32	Functions, 72
iV_GetEyeImage	Functions, 72
iV_GetFeatureKey	Functions, 73
iV_GetGazeChannelQuality	Functions, 73
iV_GetGeometryProfiles	Functions, 73
iV_GetLicenseDueDate	Functions, 74
iV_GetREDGeometry	Functions, 74
iV_GetSample	Functions, 74
iV_GetSample32	Functions, 75
iV_GetSceneVideo	Functions, 75
iV_GetSerialNumber	Functions, 76
iV_GetSpeedModes	Functions, 76
iV_GetSystemInfo	Functions, 76
iV_GetTrackingMonitor	Functions, 77
iV_GetTrackingStatus	Functions, 77
iV_HideAccuracyMonitor	Functions, 78
iV_HideEyeImageMonitor	Functions, 78
iV_HideSceneVideoMonitor	Functions, 78
iV_HideTrackingMonitor	Functions, 78
iV_IsConnected	Functions, 79
iV_LoadCalibration	Functions, 79
iV_Log	Functions, 79
iV_PauseEyetracking	Functions, 80
iV_PauseRecording	Functions, 80
iV_Quit	Functions, 80
iV_RecalibrateOnePoint	Functions, 81
iV_ReleaseAOIPort	Functions, 82
iV_RemoveAOI	Functions, 82
iV_ResetCalibrationPoints	Functions, 82
iV_SaveCalibration	Functions, 83
iV_SaveData	Functions, 83
iV_SelectREDGeometry	Functions, 84

- iV_SendCommand
 - Functions, [84](#)
- iV_SendImageMessage
 - Functions, [84](#)
- iV_SetAOIHitCallback
 - Functions, [85](#)
- iV_SetCalibrationCallback
 - Functions, [85](#)
- iV_SetConnectionTimeout
 - Functions, [86](#)
- iV_SetEventCallback
 - Functions, [86](#)
- iV_SetEventDetectionParameter
 - Functions, [87](#)
- iV_SetEyeImageCallback
 - Functions, [87](#)
- iV_SetLicense
 - Functions, [88](#)
- iV_SetLogger
 - Functions, [88](#)
- iV_SetREDGeometry
 - Functions, [88](#)
- iV_SetResolution
 - Functions, [89](#)
- iV_SetSampleCallback
 - Functions, [89](#)
- iV_SetSceneVideoCallback
 - Functions, [90](#)
- iV_SetSpeedMode
 - Functions, [90](#)
- iV_SetTrackingMonitorCallback
 - Functions, [90](#)
- iV_SetTrackingParameter
 - Functions, [91](#)
- iV_SetupCalibration
 - Functions, [91](#)
- iV_SetupDebugMode
 - Functions, [92](#)
- iV_SetupLptRecording
 - Functions, [92](#)
- iV_ShowAccuracyMonitor
 - Functions, [93](#)
- iV_ShowEyeImageMonitor
 - Functions, [93](#)
- iV_ShowSceneVideoMonitor
 - Functions, [93](#)
- iV_ShowTrackingMonitor
 - Functions, [94](#)
- iV_Start
 - Functions, [94](#)
- iV_StartRecording
 - Functions, [94](#)
- iV_StopRecording
 - Functions, [95](#)
- iV_TestTTL
 - Functions, [95](#)
- iV_Validate
 - Functions, [96](#)
- ImageStruct, [50](#)
- MRI
 - Enumerations, [42](#)
- monitorIntegrated
 - Enumerations, [43](#)
- NONE
 - Enumerations, [42](#)
- Query
 - Enumerations, [43](#)
- RED
 - Enumerations, [42](#)
- REDGeometryStruct, [51](#)
- REDm
 - Enumerations, [42](#)
- REDn
 - Enumerations, [43](#)
- REDGeometryEnum
 - Enumerations, [43](#)
- SampleStruct, [51](#)
- SampleStruct32, [52](#)
- Set
 - Enumerations, [43](#)
- SpeedModeStruct, [52](#)
- standalone
 - Enumerations, [43](#)
- SystemInfoStruct, [52](#)
- TrackingStatusStruct, [53](#)