Smart Eye AB

Programmer’s Guide

Revision 9.2

Programmer's Guide

Revision 9.2

June 10, 2021

Smart Eye AB

Första Långgatan 28B • 413 27 Gothenburg • Sweden

Phone +46 31 60 61 60

www.smarteye.se

support@smarteye.se

Chapter 1

Abbreviations

SET *Smart Eye Tracker*

SEP *Smart Eye Pro*

WCS World Coordinate System

Contents

Chapter 2

Introduction

This document describes technical details of communication protocols available in *Smart Eye Pro* (SEP), targeted at integrations between SEP and other systems. The documentation in this document assumes that the reader is already familiar with programming in C/C++ as example code and datatypes are in most cases exclusively presented in these languages. For general usage and configuration of SEP, please see the *Smart Eye Pro Manual*.

Note

The SEP protocols will be documented fully but some protocols may be partially unavailable on your system depending on configuration and license used. For details on what limitations apply to specific SEP editions see Appendix C, Limitations of Smart Eye Pro Editions.

Chapter 3

Smart Eye Data Communication

3.1 Smart Eye Packet

There exist important C/C++ header files that are useful when working with *Smart Eye* network data. The header files are located in the folder `API\include` in the installation path of SEP.

3.1.1 Packet Specification (TCP/IP, UDP/IP)

The basic structure of the data packet produced by the different versions of SEP is the same. Please see Table 3.1 for an illustration of the packet structure.

Packet header		Sync Id SEu32, 4 bytes
		Packet type SEu16, 2 bytes
		Packet length SEu16, 2 bytes
Subpacket 1	Subpacket header	Id SEu16, 2 bytes
		Length SEu16, 2 bytes
	Subpacket data	Data . . . .
Subpacket N	Subpacket header	Id SEu16, 2 bytes
		Length SEu16, 2 bytes
	Subpacket data	Data . . . .

Table 3.1: A Schematic View Over the Smart Eye Network Packet Structure

The data packet always starts with the packet header. The packet header declares the type of the packet and the size of the data included in the packet. After the header a number of sub packets follow.

The size (Packet length) declared in the packet header indirectly defines the number of sub packets. The size defines the size in bytes and not the actual number of sub packets. The number of sub packets may differ between two consecutive data packets. The client is responsible for interpreting the data packets correctly.

Each sub packet begins with a sub packet header. The sub packet header contains two pieces of information: id and size (length). The id indicates what data the sub packet contains, for example, SELeftGazeDirection or SEHeadPosition. For a full list of all possible data ids see Table A.1.

Please note that the actual data type (double, int etc.) is not indicated in the packet. The client is expected to look up the data type of the data id. The SocketClient (see Section 6.1) sample that is included with the application illustrates how this may be done.

Note

Please note that data is sent in network byte order. I.e. on an Intel based system the byte order needs to be reversed in order to interpret the data correctly. The sample client also illustrates this.

Note

In very old versions of SEP the package format was different. Those package types are still supported in the software but not documented here. The package format described here is of type 4.

3.1.2 Data Types

Table 3.2: A List of all the available Smart Eye Data Types.

DataType	**TypeID**	**Description**
SEType_u8	0x0000	Unsigned integer 1 byte
SEType_u16	0x0001	Unsigned integer 2 bytes
SEType_u32	0x0002	Unsigned integer 4 bytes
SEType_s32	0x0003	Signed integer 4 bytes
SEType_u64	0x0004	Unsigned integer 8 bytes
SEType_f64	0x0005	Floating point 8 bytes
SEType_Point2D	0x0006	SEType_float x SEType_float y
SEType_Vect2D	0x0007	SEType_float x SEType_float y
SEType_Point3D	0x0008	SEType_float x SEType_float y SEType_float z
SEType_Vect3D	0x0009	SEType_float x SEType_float y SEType_float z
SEType_String	0x000A	SEType_u16 characters (N) SEType_u8 character 1 SEType_u8 character 2 ... SEType_u8 character N
SEType_Vector SEType_Vector<T>	0x000B	SEType_u16 elements (N) SEType_u16 typeId of element 1 Element 1 SEType_u16 typeId of element 2 Element 2 ... SEType_u16 typeId of element N Element N
SEType_Struct	0x000C	SEType_u16 elements (N) SEType_String id of element 1 SEType_u16 typeId of element 1 Element 1 SEType_String id of element 2 SEType_u16 typeId of element 2 Element 2 ... SEType_String id of element N SEType_u16 typeId of element N Element N
SEType_WorldIntersection	0x000D	SEType_u16 intersections (0|1) [SEType_Point3D worldPoint] [SEType_Point3D objectPoint] [SEType_String objectName]

Table 3.2: A List of all the available Smart Eye Data Types.

DataType	TypeID	Description
SEType_WorldIntersections	0x000E	SEType_u16 intersections (N) [SEType_Point3D worldPoint1] [SEType_Point3D objectPoint1] [SEType_String objectName1] ... [SEType_Point3D worldPointN] [SEType_Point3D objectPointN] [SEType_String objectNameN]
SEType_PacketHeader	0x000F	SEType_u32 syncId SEType_u16 packetType SEType_u16 length
SEType_SubPacketHeader	0x0010	SEType_u16 id SEType_u16 length
SEType_f32	0x0011	Floating point 4 bytes
SEType_Matrix3x3	0x0012	A 3x3 Matrix
SEType_Matrix2x2	0x0013	A 2x2 Matrix
SEType_Quaternion	0x0014	SEType_float w SEType_float x SEType_float y SEType_float z
SEType_UserMarker	0x0015	SEType_u16 exists (0|1) [SEType_s32 error] [SEType_u64 timeStamp] [SEType_u64 cameraClock] [SEType_u8 cameraIdx] [SEType_u64 data]
SEType_float	N/A	Alias for either SEType_f32 or SEType_f64

3.1.3 SubPacket Ids

Please see the Appendix A for a list of the possible output data.

3.2 SEPacketAPI.h

3.2.1 findDataInPacket

For all "SEOutputDataIds" there exist a function called "findDataInPacket" that finds the correct subpacket in the body and reads the data from that subpacket. All functions have three parameters except for the functions working with the type "WorldIntersections" (note plural), which have 4 parameters.

```
bool findDataInPacket( const unsigned int& numericalId,
          const char* pPacket,
        <SEType>& value )
bool findDataInPacket( const unsigned int& numericalId,
        const char* pPacket,
        SEWorldIntersectionStruct value[],
        SEu16& numberOfIntersections );
```

Parameter	I/0	Description
numericalId	I	The SEOutputDataId of the desired data
pPacket	I	Pointer to beginning of the packet buffer
Value	0	The extracted data, can be any SEType

Parameter	I/0	Description
numericalId	I	The SEOutputDataId of SEAllWorldIntersections or SEFilteredAllWorldIntersections
pPacket	I	Pointer to beginning of the packet buffer
Value	0	The array of extracted SEType_WorldIntersections
numberOfIntersections	0	Number of objects that has been extracted

3.2.2 readValue

For all "SETypes" there exist a function called "readValue" that reads data from a packet header, subpacket header or subpacket body. All functions have three parameters except for the function working with the type WorldIntersections (note plural), which have 4 parameters.

```
void readValue( SEPoint2D& value,
                const char* data,
                int& pos);
void readValue( SEWorldIntersectionStruct value[],
                SEu16& numberOfIntersections,
                const char* data,
                int& pos );
```

Parameter	I/0	Description
value	0	The extracted data can be any SEType
data	I	Pointer to beginning of the packet buffer
pos	I/0	Position in packet buffer

Parameter	I/0	Description
value	0	The array of extracted WorldIntersections
numberOfIntersections	0	Number of objects that has been extracted
data	I	Pointer to beginning of the packet buffer
pos	I/0	Position in packet buffer

3.2.3 printPacketContents

A function that prints all data in a packet to standard out.

```
void printPacketContents( char* pPacket, const SEu16& packetType );
```

Parameter	I/0	Description
pPacket	I	Pointer to beginning of the packet buffer
packetType	I	Type of packet, found in the header

3.2.4 printSmartEyePacket4

A function that prints all data in a packet of type 4 to standard out.

```
int printSmartEyePacket4( char* pPacket );
```

Parameter	I/0	Description
pPacket	I	Pointer to beginning of the packet buffer

3.3 TCP or UDP?

In a real time context, it is strongly recommended to use UDP over TCP as the UDP socket communication is non blocking. The downside of UDP is that packets are not guaranteed to arrive, but on a local network it is practically guaranteed that they will arrive.

A logger that is not time critical can be implemented in TCP, but still it needs to be ensured that all packets are received and handled as soon as they come in. Failure to handle incoming packets will cause system lag, packet loss or even a system crash. Do not rely on the network buffer! Instead copy the incoming data to your own queues or logs as they come in.

Note

You need to handle all incoming TCP packets as soon as they are received.

Please also keep in mind that the TCP protocol itself may add an unknown and variable amount of lag. The TCP Nagle algorithm [1] is disabled in the sending end (SEP) causing data packets to be transmitted immediately instead of possibly coalescing them into larger chunks. Still there is no guarantee that this will not happen somewhere else along the way to the receiving end. For this reason TCP is not recommended for real time applications.

3.4 Real-time or delayed data streams

Some of the output values cannot be calculated in real-time and therefore will introduce a delay. If a TCP or UDP stream containing any of these outputs is created, all the outputs will be delayed. The duration of the delay depends on the nature of the specific output value. In the case of blinks for example, the stream will be delayed for about 700 ms since that is about the longest time a natural blink can take. However, the data will still be timestamped at the accurate moment in time.

If all delayed outputs in the data selection list are excluded the data will be sent in real-time. In the UDP/TCP dialog, there is a column that will show if it is real-time or not. In order to get both minimum-lag real-time data as well as outputs from the advanced filters, two separate data streams should be created. One that contains the delayed data and one that contains the real-time data.

Text logs are unaffected by this as they are not real-time logs.

3.5 Logfile specification

The log files generated by SEP are text files with ANSI encoding and tab-separated columns. The first line of the log file is the header containing all the names of the data items and all the following lines contains the values.

For data types with several components (x,y,z) the data item is separated into several columns as

$$HeadPosition.x \rightarrow HeadPosition.y \rightarrow HeadPosition.z.$$

For data types that are a vector it is seperated with an appending $\#i$ where i is the element position of the vector.

[1] http://en.wikipedia.org/wiki/Nagle's_algorithm

A complex example is the *AllWorldIntersections* data type. It is a vector of *WorldIntersection* which is a struct containing two 3D-points *worldPoint* and *objectPoint* and a string *objectName*. It is written in the log file like this:

$AllWorldIntersections\#0.worldPoint.x$	$\rightarrow$	$AllWorldIntersections\#0.worldPoint.y$	$\rightarrow$	$AllWorldIntersections\#0.worldPoint.z$	$\rightarrow$
$AllWorldIntersections\#0.objectPoint.x$	$\rightarrow$	$AllWorldIntersections\#0.objectPoint.y$	$\rightarrow$	$AllWorldIntersections\#0.objectPoint.z$	$\rightarrow$
$AllWorldIntersections\#0.objectName$	$\rightarrow$	$AllWorldIntersections\#1.worldPoint.x$	$\rightarrow$	$AllWorldIntersections\#1.worldPoint.y$	$\rightarrow$
$AllWorldIntersections\#1.worldPoint.z$	$\rightarrow$	$AllWorldIntersections\#1.objectPoint.x$	$\rightarrow$	$AllWorldIntersections\#1.objectPoint.y$	$\rightarrow$
$AllWorldIntersections\#1.objectPoint.z$	$\rightarrow$	$AllWorldIntersections\#1.objectName$	$\rightarrow$	$AllWorldIntersections\#2.worldPoint.x$	$\rightarrow$
$AllWorldIntersections\#2.worldPoint.y$	$\rightarrow$	$AllWorldIntersections\#2.worldPoint.z$	$\rightarrow$	$AllWorldIntersections\#2.objectPoint.x$	$\rightarrow$
$AllWorldIntersections\#2.objectPoint.y$	$\rightarrow$	$AllWorldIntersections\#2.objectPoint.z$	$\rightarrow$	$AllWorldIntersections\#2.objectName$	

Note

Be aware! Some data items can get the value null, this will mean that a double tab will be written to the log file like this "$\backslash t \backslash t$" ($\rightarrow\rightarrow$). Some tokenizing/splitting functions in different programming languages will skip this empty element and a mismatch in the number of columns compared to the header will occur.

Note

Some values will be written with scientific notations, e.g. 9.52913606973326e-005.

Note

For floating point values a precision of 15 digits is used.

3.6 A detailed look at some output data

Note

It is important to consider the quality value for any output data used. A low quality value may indicate that the associated output data is inaccurate. The quality values are those values whose names end with a Q.

3.6.1 Head Position Quality

The head position quality can take 3 values, a value of 1.0 denotes that the head is found and tracked in at least 2 cameras, while the value 0.5 means that only one camera is tracking. The value 0.0 means that no face is found at all.

Generally speaking, a recommendation is to only use head data with a quality value of 1.0.

3.6.2 Gaze Direction Quality

The GazeDirectionQ value can take any floating point value between 0.0 and 1.0. It depends on the placement of cameras and flashes, the tracking algorithms used and each individuals eye features. This means that it is not easy to set a general threshold, for example 0.3, where all gaze data below that threshold should be considered unreliable, for all possible setups and subjects.

However, it is to tedious for the user to change this threshold for each specific individual during analysis. So in order to be able to use some generic threshold an attempt is made to normalize this value to the range [0.0, 1.0]. The normalized value 0.0 corresponds to the 1st percentile of all collected quality values of the current tracking session while 1.0 corresponds to the 99th percentile respectively. It takes some samples with good and bad quality measurements before the normalization has built up enough statistics to be effective. Therefore the quality value of the first few seconds of a tracking session may not be comparable to the succeeding quality values.

Note

Normalization is done individually for each camera. This means that the quality levels from the different cameras may differ slightly, especially in the beginning of a tracking session.

The quality value that the system outputs is the one of the camera that is expected to have the best view of the eye. There is no averaging between the cameras. When the system thinks another camera has a better view of the eye, that camera's quality value is the output. Consequently, if the cameras have different normalized quality levels, there might be different quality levels put out of the system.

Normally, this effect is only significant at the very beginning of a tracking session. However, if the system is configured in a way that one of the cameras is used very little, there might be a risk that this camera will provide quality levels in a different range than the other cameras.

A procedure that can be used to avoid this problem is to start each tracking session by looking into each camera once and blink the eyes. This will fill up the normalization buffer with enough statistics.

The qaze direction quality depends on whether the gaze is calculated from pupil or iris detections and uses the quality of the respective detections. In case of pupil-based gaze calculation, this quality is related to how well the detected pupil resembles an ellipse shape and is consequently decreased by disturbances in the pupil edge, such as glints or eyelids.

The iris-based gaze calculation is more complicated. The gaze direction quality reflects the strength of the edge between iris and sclera. This value highly depends on the eye color, illumination and other factors. Therefore the range of this value is more individual.

3.6.3 Gaze Ray, Gaze Origin and Gaze Direction

A "Gaze Ray" is defined by a "Gaze Origin" point in 3D space plus a "Gaze Direction" vector. The "Gaze Direction" vector is a unit vector representing a direction in 3D space.

There is a separate gaze ray for the left and the right eye. These can in turn be combined into a "Cyclops" gaze ray for a virtual eye positioned at the midpoint between the left and the right eye.

Note that no explicit information about the distance to the gaze target is contained in the gaze rays. By estimating the intersection of the left and right gaze rays, some distance information may be recovered, but it will not be very accurate for large distances.

3.6.4 Blinks

The system has a filter that detects blinks by evaluating the measured eyelid opening samples over a period of approximately 700 ms. This means that blinks that last longer than 700 ms will not be considered as blinks.

All samples that belong to a blink will be marked with blink id in the output data. The blink id is an integer that is incremented for each detected blink. Samples that belong to the same blink occurrence will have the same blink id. When no blink is detected, the blink id value is zero. The blink duration may be calculated as the time difference between last and first samples with the same blink id.

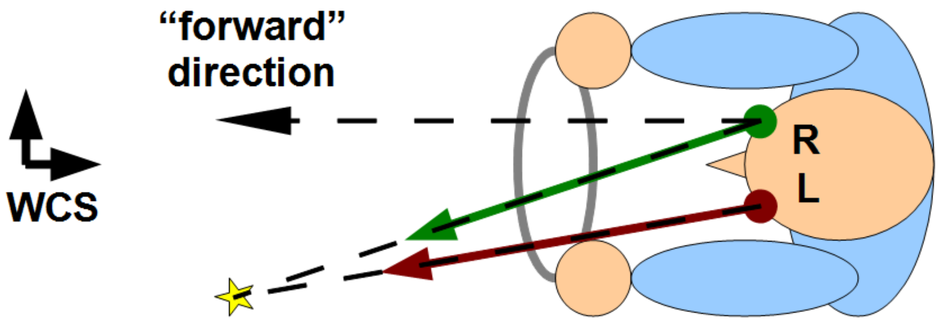

Figure 3.1: Gaze Rays for the left (red) and right (green) eye for a top view of a driver looking at a gaze target (symbolised by a yellow star). The 3D (X, Y, Z relative to WCS) Gaze Origin points are drawn as round dots and the unit-length Gaze Direction vectors are drawn as arrows.

As this filter evaluates samples over a period of time, it will cause a delay in the data stream, for which blink data is selected. See section 3.4 on how to get both real time and delayed data.

Note

Blink output creates a lag, so all other data in an output stream that contain blink data will also lag. For a real time data stream this value should be excluded.

The characteristics of a blink can look like the diagram below. Normally the measured eyelid opening does not go all the way down to 0 mm, thus the dashed line just above the time axis.

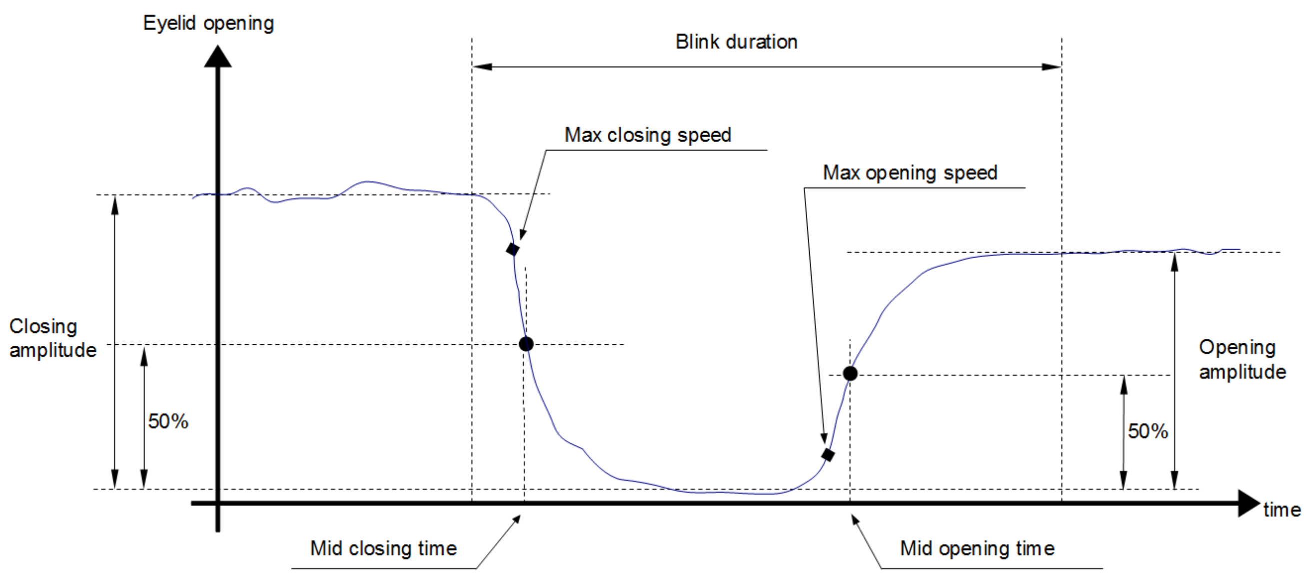

Figure 3.2: Blink characteristics and measured values

The system measures the characteristics of a blink. The "Closing amplitude" is the difference in eyelid opening before the closing movement and while the eye is closed. The "Opening amplitude" is the difference between the eyelid opening for closed eye and the eyelid opening just after the opening movement. Opening and closing amplitude are not necessarily equal.

"Mid Closing time" is the point in time when the eye is half way down during the closing movement. More precisely it is the time when the eyelid opening has reached 50% of the closing amplitude. To find this time the system fits a typical eyelid movement curve to the measured eyelid opening samples. The exact time is then given by finding the point on this curve where it reaches 50% of the amplitude. "Mid Opening time" is determined similarly. Both "Mid Closing time" and "Mid Opening time" are reported according to the UserTimeStamp output if a User Time dll has been specified. Otherwise they are reported according to the RealTimeClock output.

"Max closing speed" is the maximum speed of the closing movement, calculated from the fitted curve. Analogously for "Max opening speed".

The values describing blink characteristics are only non-zero at the first sample of each blink occurrence.

While the "blink id" output data is a consensus measurement, that is based on both eyes' eyelid behavior, the values describing the characteristic of the eyelid movement are determined and sent out for each eye separately.

3.6.5 Saccades and fixations

There is also a filter to detect saccades and fixations. The output of these fields are similar to the blink output. The output is an integer that will increment for each saccade or fixation. I.e. it will be a zero when no saccade is occurring and a non-zero integer will indicate that the n:th saccade is occurring. The fixation value will increment after each blink. This output causes a delay since it need to look at future frames to determine if a saccade or a fixation is occurring or not.

Note

Saccades and fixations output creates a lag, so all other data in an output stream will be delayed. For a real time data stream this value needs to be excluded.

For more information on how to adjust the settings for these filters, read the manual for the *Smart Eye* software that you are working with.

3.7 CAN Output Specification

The *Smart Eye* CAN Output interface uses 31 predefined packets containing much of the same information as the data output specification.

All packets have a numerical id (nID in the table below) that is added to a user defined message base for use on the CAN bus. The default message base is 0x400. The message definitions can be found in the `SECANMsgTypes.h` in the *Smart Eye* API as well as in the CANCaseXL-client sample.

The packet data is normally 8 bytes long, except for the SE_CAN_EYECLOSURE and SE_CAN_PUPILDIAMETER packets which are 4 bytes each.

To save bandwidth all values (except Framenumber and UserTimeStamp) are multiplied by 10 000 before they are sent, which means that metric values will be in 1/10 mm, time measurement in 1/10 ms, and angular values in 1/10 000 radians. Quality values are between 1 and 10000. Most values will be in the form of 16 bit integers, except for Framenumber which has to be a 32 bit integer and the user defined Timestamp which is a 64 bit integer.

On the CAN bus, little endian is used, i.e the least significant byte is always sent first for multiple byte values.

Some messages are sent with a longer delay due to the nature of the data. This group of packages is called "non-realtime messages" and consists of all messages between and including SE_CAN_SYNCH_NON_REALTIME to SE_CAN_RIGHTBLINKOPENINGSPEED.

Note

Output Data is always sent as little endian on the CAN bus.

Table 3.3: CAN Output Data Specification

Packet ID	nID	Data Byte Type ID	Unit	Description
SE_CAN_SYNCH	1	[0..3] u32 Framenumber [4..5] u16 Timestamp [6..7] u16 EstimatedDelay	 1/10ms 1/10ms	The time stamp will wrap around at 6.5536 s and the estimated delay will saturate at 6.5535s
SE_CAN_USERTIMESTAMP	2	[0..7] u64 UserTimeStamp	User Defined	Used to synchronize Smart Eye Pro with external events.
SE_CAN_HEADPOSITION	3	[0..1] s16 HeadPosition.x [2..3] s16 HeadPosition.y [4..5] s16 HeadPosition.z [6..7] u16 HeadPositionQ	1/10mm 1/10mm 1/10mm 0-10000	The head position, scaled down to 1/10 mm.
SE_CAN_HEADROTATION	4	[0..1] s16 HeadRotation.x [2..3] s16 HeadRotation.y [4..5] s16 HeadRotation.z [6..7] u16 HeadRotationQ	1/10000 rad 1/10000 rad 1/10000 rad 0-10000	The head position in rodrigues format, scaled down to 1/10000 radians.
SE_CAN_GAZEORIGIN	5	[0..1] s16 GazeOrigin.x [2..3] s16 GazeOrigin.y [4..5] s16 GazeOrigin.z [6..7] u16 GazeOriginQ	1/10mm 1/10mm 1/10mm 0-10000	The origin of the gaze vector, scaled down to 1/10 mm.
SE_CAN_GAZEDIRECTION	6	[0..1] s16 GazeDirection.x [2..3] s16 GazeDirection.y [4..5] s16 GazeDirection.z [6..7] u16 GazeDirectionQ	 0-10000	The unit vector gaze direction, but between -10000 and 10000.
SE_CAN_EYECLOSURE	7	[0..1] s16 EyeLidOpening [2..3] u16 EyeLidOpeningQ	1/10mm 0-10000	The average eye lid opening value.
SE_CAN_BOTHEYE CLOSURES	8	[0..1] s16 LeftEyeLidOpening [2..3] U16 LeftEyeLidOpeningQ [4..5] s16 RightEyeLidOpening [6..7] u16 RightEyeLidOpeningQ	1/10mm 0-10000 1/10mm 0-10000	Both the left and right eye lid opening values with quality values.
SE_CAN_PUPILDIAMETER	9	[0..1] s16 PupilDiameter [2..3] u16 PupildiameterQ	1/10mm 0-10000	The diameter of the pupil in 1/10 mm scale.
SE_CAN_BOTHPUPIL DIAMETERS	10	[0..1] s16 LeftPupildiameter [2..3] u16 LeftPupildiameterQ [4..5] s16 RightPupildiameter [6..7] u16 RightPupildiameterQ	1/10mm 0-10000 1/10mm 0-10000	The diameter of the pupil in 1/10 mm scale for both the left and the right eye.
SE_CAN_ASCIIKEY	11	[0] u8 ASCIIKeyboardState		Last keypress value in ASCII format.
SE_CAN_WORLD INTERSECTION	12	[0..1] u16 ZoneID [2..3] s16 ObjectPoint.x [4..5] s16 ObjectPoint.y [6..7] s16 ObjectPoint.z	0-65535 1/10mm 1/10mm 1/10mm	Intersection of a World object. The zone ID and the intersected position are reported.
SE_CAN_FILTERED GAZEDIRECTION	13	[0..1] s16 FilteredGazeDirection.x [2..3] s16 FilteredGazeDirection.y [4..5] s16 FilteredGazeDirection.z		Filtred gaze direction in rodrigues format and a quality value

Table 3.3: CAN Output Data Specification

Packet ID	nID	Data Byte Type ID	Unit	Description
		[6..7] u16 FilteredGazeDirectionQ	0-10000	between 0 and 10000.
SE_CAN_FILTERED LEFTGAZEDIRECTION	14	[0..1] s16 FilteredLeftGazeDirection.x [2..3] s16 FilteredLeftGazeDirection.y [4..5] s16 FilteredLeftGazeDirection.z [6..7] u16 FilteredLeftGazeDirectionQ	 0-10000	Filtred gaze direction for the left eye in rodrigues format with a quality value.
SE_CAN_FILTERED RIGHTGAZEDIRECTION	15	[0..1] s16 FilteredRightGazeDirection.x [2..3] s16 FilteredRightGazeDirection.y [4..5] s16 FilteredRightGazeDirection.z [6..7] u16 FilteredRightGazeDirectionQ	 0-10000	Filtered gaze direction for the right eye in rodrigues format with a quality value.
SE_CAN_SYNCH NON_REALTIME	16	[0..3] u32 Framenumber [4..5] u16 Timestamp [6..7] u16 EstimatedDelay	 1/10ms 1/10ms	Same as nID 1 but for non-realtime data
SE_CAN_USERTIMESTAMP NON_REALTIME	17	[0..7] u64 UserTimeStamp	User Defined	Sane as nID 2 but for non-realtime data
SE_CAN_BLINK	18	[0..3] u32 Blink		When the blink detection filter detects a blink, this value will be a non-zero integer for the duration of the blink. This integer increases for each blink and can be seen as a unique blink identifier.
SE_CAN_SACCADE	19	[0..3] u32 Saccade		When the saccade detection filter detects a saccade, this value will be a non-zero integer for the duration of the saccade. This integer increases for each saccade and can be seen as a unique saccade identifier.
SE_CAN_LEFTBLINK CLOSINGMIDTIME	20	[0..7] u64 LeftBlinkClosingMidTime	100ns	See Section 3.6.4.
SE_CAN_LEFTBLINK CLOSINGAMPLITUDE	21	[0..1] s16 LeftBlinkClosingAmplitude	1/10mm	See Section 3.6.4.
SE_CAN_LEFTBLINK CLOSINGSPEED	22	[0..1] s16 LeftBlinkClosingSpeed	1/10mm/s	See Section 3.6.4.
SE_CAN_LEFTBLINK OPENINGMIDTIME	23	[0..7] u64 LeftBlinkOpeningMidTime	100ns	See Section 3.6.4.
SE_CAN_LEFTBLINK OPENINGAMPLITUDE	24	[0..1] s16 LeftBlinkOpeningAmplitude	1/10mm	See Section 3.6.4.
SE_CAN_LEFTBLINK OPENINGSPEED	25	[0..1] s16 LeftBlinkOpeningSpeed	1/10mm/s	See Section 3.6.4.
SE_CAN_RIGHTBLINK CLOSINGMIDTIME	26	[0..7] u64 RightBlinkClosingMidTime	100ns	See Section 3.6.4.
SE_CAN_RIGHTBLINK CLOSINGAMPLITUDE	27	[0..1] s16 RightBlinkClosingAmplitude	1/10mm	See Section 3.6.4.
SE_CAN_RIGHTBLINK CLOSINGSPEED	28	[0..1] s16 RightBlinkClosingSpeed	1/10mm/s	See Section 3.6.4.
SE_CAN_RIGHTBLINK	29	[0..7] u64 RightBlinkOpeningMidTime	100ns	See Section 3.6.4.

Table 3.3: CAN Output Data Specification

Packet ID	nID	Data Byte Type ID	Unit	Description
OPENINGMIDTIME				
SE_CAN_RIGHTBLINK OPENINGAMPLITUDE	30	[0..1] s16 RightBlinkOpeningAmplitude	1/10mm	See Section 3.6.4.
SE_CAN_RIGHTBLINK OPENINGSPEED	31	[0..1] s16 RightBlinkOpeningSpeed	1/10mm/s	See Section 3.6.4.

For further details refer to the .dbc file in the `"Doc/dbc"` folder under the installation path.

3.7.1 World Intersections

To get a numerical value as the id of the object the gaze interesects, we use any numerical values in the current closest world intersection object name. To get values for all different intersections, make sure to add a unique numerical value to all Planes, Zones, Spheres and Boxes in the world model. *"Zone222"* will for example give the zone id 222 and *"111Zone333"* will give a zone id of 111333.

The object positions are from the object position value of the Closest World Intersection, but with the unit 1/10 mm.

Chapter 4

Smart Eye Remote Control

The optional module remote control interface allows you to control SEP from an external client via an JSON-RPC interface. Since this interface is following a open and public specification you can control SEP from any computer using any OS and programming language of your choice.

4.1 JSON-RPC

4.1.1 JSON-RPC 2.0 specification

Smart Eye follows the JSON-RPC 2.0 specifications which can be found here http://json-rpc.org/. A short summary will be given here.

Request

This is the requests that is being sent from the client to the SEP JSON-RPC server.

```
{
  "jsonrpc" : "2.0",                          //REQUIRED
  "method"  : <string>,                       //REQUIRED
  "params"  : <json array> or <json object>,  //OPTIONAL
  "id"      : <integer> or <string>           //REQUIRED
}
```

Response

To every request a response is given from the server to the client.

```
{
  "jsonrpc" : "2.0",                  //REQUIRED
  "result"  : <any json primitive>,   //REQUIRED on SUCCESS, MUST NOT EXIST on ERROR
  "error"   : <error object>,         //REQUIRED on ERROR, MUST NOT EXIST on SUCCESS
  "id"      : <integer> or <string>   //REQUIRED
}
```

Notification

A notification is a request that shall NOT get a response.

```
{
  "jsonrpc" : "2.0",                            //REQUIRED
  "method"  : <string>,                         //REQUIRED
  "params"  : <json array> or <json object>, //OPTIONAL
}
```

Error object

If an error occurs, this is the format of the error object.

```
{
  "code"    : <integer>               //REQUIRED
  "message" : <string>,               //REQUIRED, SHOULD be one single sentence
  "data"    : <any json primitive>    //OPTIONAL, any additional data that server
                                      //may want to attach with the error.
}
```

Pre-defined error codes

These are predefined by the JSON-RPC specification.

CODE	HEX	MESSAGE	MEANING
-32700	0xFFFF8044	Parse error	Invalid JSON was received by the server. An error occurred on the server while parsing the JSON text.
-32600	0xFFFF80A8	Invalid request	The JSON sent is not a valid Request object.
-32601	0xFFFF80A7	Method not found	The method does not exist / is not available.
-32602	0xFFFF80A6	Invalid params	Invalid method parameter(s).
-32603	0xFFFF80A5	Internal error	Internal JSON-RPC error.
-32000 to -32099	0xFFFF8300 to 0xFFFF829D	Server error	Reserved for implementation-defined server-errors.

4.1.2 Connecting to the the JSON-RPC Server

The server is started automatically together with SEP assuming that the module *Remote Control* is part of the license. It listens for TCP connections on port 8100 by default, but can be changed through the Windows Registry settings.

Each connected client receives a unique id, so it is possible to have multiple connected clients to the Smart Eye JSON-RPC server. The server can also act as a communication hub allowing for connected clients to send notifications between each other.

4.1.3 Transportation layer

The JSON-RPC 2.0 specification says nothing about the transportation layer. That means that the transportation layer is up to the implementor to choose and can therefore vary between different JSON-RPC libraries.

As mentioned in the previous section SEP is expecting a TCP connection. That is one part of the transportation layer. But since TCP is a streaming protocol there is no sure way to seperate between the JSON messages if they are just sent as pure strings. That is why SEP is expecting the JSON requests and responses to be encoded as a *netstring*.

What netstring encoding does is simply to add the length of the expected string before it together with a ":" and ends the string with a ",", as *"5:smart,3:eye,"*. The full specification can be found here `https://cr.yp.to/proto/netstrings.txt`.

4.1.4 RPC Timings

Sending lots of RPC calls in short order can sometimes cause dead locks in SEP. If you are experiencing this, as a workaround try adding a one second delay between your RPC calls.

4.1.5 Example DLL

One of the ways to communicate with the JSON-RPC server (SEP) is to use the native-c dll named `"SEJsonRpc.dll"`. The files can be found under `API\example\bin\` within the installation folder of SEP. The communication between client and server is initialized using the function **seInitialize** and is freed using **seFree**. Refer to the header file and to the commands in Section 4.1.6 for information about using the dll. A C++ example application using the SEJsonRpc.dll for RPC communication is also available, see Section 6.6.

4.1.6 JSON-RPC Methods

The following methods are available on the SEP JSON-RPC server.

ping
getRealTimeClock
getRPCVersion
getProductName
getCameraType
getFirmwareVersions
getState
getRecordingState
startTracking
stopTracking
startChessboardTracking
stopChessboardTracking
setLogSpecification
setLogFile
startLog
stopLog
setRecordingFile
startRecording
stopRecording
setImageSourceCameras
setImageSourceRecording
setPlaybackSpeedToMax
setPlaybackSpeedToRealTime
setPlaybackPosition
setPlaybackStartStopPositions
resumePlayback
pausePlayback
setPlaybackRepeatOn
setPlaybackRepeatOff
startCollectPointSamplesAutomatic
stopCollectPointSamplesAutomatic
clearProfile
saveProfile
loadProfile
loadWorldModel
getWorldModel
setWorldModel
openDataStreamUDP
closeDataStreamUDP
openDataStreamTCP
closeDataStreamTCP
startCollectSamplesWCS
startCollectSamplesObject
stopCollectSamples
clearAllTargetSamples
clearTargetSamples
retrieveTargetStatistics
retrieveCalibrationResults
calibrateGaze
applyGazeCalibration
clearGazeCalibration
shutdown
keyDown
keyUp
subscribeToNotification
unsubscribeToNotification
sendNotification
getCameraImage
isCameraConnectedToUSB3

Keep alive

ping

Description:
Sends a ping to the server which replies with a pong. This method call will respond as soon as possible, even though the request queue is not empty or there is an ongoing procedure on the server side. This method is useful to call in a keep-alive polling system.

Result:
[pong] <string> The ping response.

Request Example:

```
{"jsonrpc":"2.0", "method":"ping", "id":0}
```

Response Example:

```
{"jsonrpc":"2.0","result":"pong","id":0}
```

Time synchronization

getRealTimeClock

Description:
Returns the local computer clock represented as a 64-bit value in FILETIME format as an absolute time since January 1, 1601 (UTC).

Result:

[RealTimeClock]	<string>	Current real time clock in FILETIME format.

Request Example:

```
{"jsonrpc":"2.0", "method":"getRealTimeClock", "id":0}
```

Response Example:

```
{"jsonrpc":"2.0", "result":"132282210320546339", "id":0}
```

Server json-rpc interface version

Use this method to assure compability between different versions of client applications and SEP.

getRPCVersion

Description:
Gets the version of the SEP JSON-RPC interface.

Result:

[major]	<int>	Major version.
[minor]	<int>	Minor version.

Request Example:

```
{"jsonrpc":"2.0", "method":"getRPCVersion", "id":0}
```

Response Example:

```
{"jsonrpc":"2.0","result":{"major":1, "minor":0},"id":0}
```

getProductName

Description:
Returns a user friendly product name.

Result:
[productName] <string> The product name.

Request Example:

```
{"jsonrpc":"2.0", "method":"getProductName", "id":0}
```

Response Example:

```
{"jsonrpc":"2.0","result":"Smart Eye Pro 7.2","id":0}
```

getCameraType

Description:
Returns the live camera type.

Result:
[cameraType] <string> The camera type.

Request Example:

```
{"jsonrpc":"2.0", "method":"getCameraType", "id":0}
```

Response Example:

```
{"jsonrpc":"2.0","result":"Aurora","id":0}
```

getFirmwareVersions

Description:
Returns an array of firmware versions of the currently connected cameras.

Result:
[firmwareVersions] <string array> Array of firmware versions. **Note:** Availability and response format differs between camera models.

Request Example:

```
{"jsonrpc":"2.0", "method":"getFirmwareVersions", "id":0}
```

Response Example:

```
{"jsonrpc":"2.0","result":["FW: 5.1, FPGA: 5.2","FW: 5.1, FPGA: 5.2"],"id":0}
```

Tracker States

The eye tracker can be in different modes/states. The full list of possible states are found in the table below.

Name	Hex	Description
Idling	0x0000	The system is idle, ready to go into any other state.
Tracking	0x0010	The system is tracking
ChessboardTracker	0x0015	The system is currently tracking a chessboard
CalibratingCameras	0x0020	The system is currently performing camera calibration
VerifyingCalibration	0x0021	The system is verifying the camera calibration
CheckingFocus	0x0030	The system is adjusting focus and aperture
Mtf	0x0031	The system is performing a MTF measurements
DefiningWCS	0x0040	The system is performing manual WCS definition
DefiningAutomaticWCS	0x0041	The system is in an defining WCS automatically using a chessboard
DefiningCameraTiedWCS	0x0042	The system is defining wcs manually, but using cameras for position and rotation
GazeCalibrating	0x0050	The system is performing gaze calibration
GazeVerifying	0x0051	The system is verifying gaze calibration
RecordingProfile	0x0060	The system is collecting snapshots for profile creation
PlacingMarkers	0x0061	The system is placing markers in profile creation
ProfileSelectingPoses	0x0062	The system is selecting/reviewing/removing poses
ProfileReselectingPoses	0x0063	The system is reselecting poses
RecordingToFile	0x0070	[OBSOLETE] The system is recording to disk
ScriptedRecordingToFile	0x0071	[OBSOLETE] The system is recording to disk (without open UI dialog)
DefiningExpectedPose	0x0080	The system is defining expected pose
WorldMeasurement	0x0090	[OBSOLETE] The system is measuring/building a world model
AutoDetectLens	0x00A0	The system is detecting lenses

getState

Description:
Gets the SEP state.

Result:
[state] <int> The current state.

Request Example:

```
{"jsonrpc":"2.0", "method":"getState", "id":0}
```

Response Example:

```
{"jsonrpc":"2.0", "result":{"state":4}, "id":0}
```

Record States

The full list of possible recordings states is found in the table below.

Name	Hex	Description
NotRecording	0x1000	The system is not currently recording.
Recording	0x1010	The system is currently recording.

getRecordingState

Description:
Gets the SEP recording state.

Result:
[recordingState] <int> The current recording state.

Request Example:

```
{"jsonrpc":"2.0", "method":"getRecordingState", "id":0}
```

Response Example:

```
{"jsonrpc":"2.0", "result":{"recordingState":4096}, "id":0}
```

Tracking

startTracking

Description:
Start tracking mode.

Request Example:

```
{"jsonrpc":"2.0", "method":"startTracking", "id":0}
```

stopTracking

Description:
Stop tracking mode.

Request Example:

```
{"jsonrpc":"2.0", "method":"stopTracking", "id":0}
```

startChessboardTracking

Description:
Start the chessboard tracking mode

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "startChessboardTracking",
  "id"      : 0
}
```

stopChessboardTracking

Description:
Stop the chessboard tracking mode

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "stopChessboardTracking",
  "id"      : 0
}
```

Logging

setLogSpecification

Description:
Sets the output data specification for text logging.

Parameters:
[0] <string> The output data specification, seperated by ';'. See full list of available output data in the Appendix B.1.

Request Example:

```
{"jsonrpc":"2.0", "method":"setLogSpecification", "params":["ClosestWorldIntersection;
    GazeDirectionQ"], "id":0}
```

setLogFile

Description:
Sets the file path of the output log file.

Parameters:
[0] <string> File path to the specified output log file. The path must be reachable by the server computer.

Request Example:

```
{"jsonrpc":"2.0", "method":"setLogFile", "params":["d:\\logs\\my_log.log"], "id":0}
```

startLog

Description:
Starts logging a text log.
Note! This command assumes that the log path has already been set by the command ***setLogFile***

Request Example:

```
{"jsonrpc":"2.0", "method":"startLog", "id":0}
```

stopLog

Description:
Stops logging the active text log.

Request Example:

```
{"jsonrpc":"2.0", "method":"stopLog", "id":0}
```

Recording

setRecordingFile

Description:
Sets the destination to where recorded data will be saved.
Note! If the destination hard drive is not fast enough or full the recording will fail.

Parameters:

[0]	<string>	File path to the specified output recording file. The path must be reachable by the server computer.

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "setRecordingFile",
  "params"  : ["d:\\recordings\\my_recording.sma"],
  "id" : 0
}
```

startRecording

Description:
Starts recording a movie. Returns the remaining recording time (in seconds) calculated as a function of the data rate of the camera system and the available space on the recording disk.
Note! This command assumes that the recording file path has already been set by the command ***setRecordingFile***

Parameters:

[0]	<int>	[OPTIONAL] Compression parameter, can be omitted. Use "0" to turn the compression off and "2" to turn it on (default). The value "1" is a legacy compression type which is no longer supported.

Request Example:

```
{"jsonrpc":"2.0", "method":"startRecording", "id":0}
```

Request Example:

```
{"jsonrpc":"2.0", "method":"startRecording", "params":[0], "id":0}
```

Response Example:

```
{"jsonrpc":"2.0", "result":{"remainingRecordingTime":12}, "id":0}
```

stopRecording

Description:
Stops recording a movie.

Request Example:

```
{"jsonrpc":"2.0", "method":"stopRecording", "id":0}
```

Image source

setImageSourceCameras

Description:
Sets live cameras as the image source.

Request Example:

```
{"jsonrpc":"2.0", "method":"setImageSourceCameras", "id":0}
```

setImageSourceRecording

Description:
Sets a recording as the image source.

Parameters:
[0] <string> File path to the specified recording file that should be played.

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "setImageSourceRecording",
  "params"  : ["d:\\recordings\\my_recording.sma"],
  "id"      : 0
}
```

Playback

setPlaybackSpeedToMax

Description:
Sets the recording playback speed to max (as fast as the computer can handle).

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "setPlaybackSpeedToMax",
  "id"      : 0
}
```

setPlaybackSpeedToRealTime

Description:
Sets the recording playback speed to real time (the same frame rate as it was recorded in).

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "setPlaybackSpeedToRealTime",
  "id"      : 0
}
```

setPlaybackPosition

Description:
Set position of the playback. Positions starts at 0 for the first frame, and is incremented by one for each subsequent frame in the recording.

Note! The position of a frame is generally not equal to its FrameNumber.

Parameters:

[position]	<int>	Which zero-indexed position to play from.

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "setPlaybackPosition",
  "params"  : [2],
  "id"      : 0
}
```

setPlaybackStartStopPositions

Description:
Set start and stop positions of the playback. Positions starts at 0 for the first frame, and is incremented by one for each subsequent frame in the recording.

Note! The position of a frame is generally not equal to its FrameNumber.

Parameters:

[start]	<int>	Which zero-indexed position the playback should start from
[stop]	<int>	Which zero-indexed position the playback should stop at

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "setPlaybackStartStopPositions",
  "params"  : [1, 2],
  "id"      : 0
}
```

resumePlayback

Description:
Resume the playback

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "resumePlayback ",
  "id"      : 0
}
```

pausePlayback

Description:
Pause the playback

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "pausePlayback",
  "id"      : 0
}
```

setPlaybackRepeatOn

Description:
Set playback to repeat when finished playing

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "setPlaybackRepeatOn",
  "id"      : 0
}
```

setPlaybackRepeatOff

Description:
Set playback to not repeat when finished playing

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "setPlaybackRepeatOff",
  "id"      : 0
}
```

startCollectPointSamplesAutomatic

Description:
Orders SEP to start automatic point samples collection

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "startCollectPointSamplesAutomatic",
  "id"      : 0
}
```

stopCollectPointSamplesAutomatic

Description:
Orders SEP to stop automatic point samples collection

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "stopCollectPointSamplesAutomatic",
  "id"      : 0
}
```

Profile

clearProfile

Description:
Clears the current profile.

Request Example:

```
{"jsonrpc":"2.0", "method":"clearProfile", "id":0}
```

saveProfile

Description:
Saves a Profile to the specified path.

Parameters:

[0]	<string>	File path to the specified profile. The path must be reachable by the server computer. **Note!** Don't specify the file ending in the path, SEP will append that automatically. Depending on which profile mode SEP is running in the file ending will be different.

Request Example:

```
{"jsonrpc":"2.0", "method":"saveProfile", "params":["d:\\profiles\\my_profile"], "id":0}
```

loadProfile

Description:
Load a Profile with the specified path.

Parameters:
[0] <string> File path to the specified profile. The path must be reachable by the server computer.

Request Example:

```
{"jsonrpc":"2.0", "method":"loadProfile", "params":["d:\\profiles\\my_profile.hm4"], "id":0}
```

World model

loadWorldModel

Description:
Loads a World Model with the specified path, or the default world model if no path is specified.

Parameters:
[0] <string> [OPTIONAL] File path to a World Model (.sew file). The path must be reachable by the server computer.

Request Example:

```
{"jsonrpc":"2.0", "method":"loadWorldModel", "params":["d:\\world_models\\my_world_model.sew"
    ], "id":0}
```

getWorldModel

Description:
Gets the World Model currently loaded in SEP.

Result:
[worldModel] <string> The loaded World Model of SEP as a string.

Request Example:

```
{"jsonrpc":"2.0", "method":"getWorldModel", "id":0}
```

Response Example:

```
{
  "jsonrpc" : "2.0",
  "result"  : {
    "worldModel" : "Screen : {\n  name = \"MyScreen\"\n  lowerMiddle = 0.0, 0.05, -0.05\n
        size = 0.52, 0.326\n  resolution = 1920, 1200\n  calibrationPoints = 4\n}"
  },
  "id" : 0
}
```

setWorldModel

Description:
Sets the World Model in SEP.

Parameters:

[0]	<string>	A *Smart Eye* World Model.

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "setWorldModel",
  "params"  : ["Screen : {\n  name = \"MyScreen\"\n  lowerMiddle = 0.0, 0.05, -0.05\n  size =
       0.52, 0.326\n  resolution = 1920, 1200\n  calibrationPoints = 4\n}"],
  "id" : 0
}
```

Communication UDP/TCP

openDataStreamUDP

Description:
Orders SEP to open up a UDP data stream.

Parameters:

[0]	<string>	The destination IP adress. If empty "" it will default to the requester's IP adress.
[1]	<int>	The port.
[2]	<string>	The output data specification, seperated by ';'. See full list of available output data in the Appendix B.1.

Request Example:

```
{"jsonrpc":"2.0", "method":"openDataStreamUDP", "params":["192.168.0.3", 5100, "GazeDirection
    ;GazeDirectionQ"], "id":0}
```

closeDataStreamUDP

Description:
Orders SEP to close a specific UDP data stream.

Parameters:

[0]	<string>	The destination IP adress. If empty "" it will default to the requester's IP adress.
[1]	<int>	The port.

Request Example:

```
{"jsonrpc":"2.0", "method":"closeDataStreamUDP", "params":["192.168.0.3", 5100], "id":0}
```

openDataStreamTCP

Description:
Orders SEP to open up a TCP data stream.

Parameters:

[0]	<int>	The port.
[1]	<string>	The output data specification, seperated by ';'. See full list of available output data in the Appendix B.1.

Request Example:

```
{"jsonrpc":"2.0", "method":"openDataStreamTCP", "params":[5100, "GazeDirection;GazeDirectionQ
    "], "id":0}
```

closeDataStreamTCP

Description:
Orders SEP to close a specific TCP data stream.

Parameters:

[0]	<int>	The port.

Request Example:

```
{"jsonrpc":"2.0", "method":"closeDataStreamTCP", "params":[5100], "id":0}
```

Gaze Calibration

The methods in this section are used to perform a gaze calibration procedure remotely. A pseudocode gaze calibration procedure could look like this

```
clearAllTargetSamples;
foreach calibration points do
    display calibration point;
    startCollectSamplesWCS;
    sleep for 1000 ms;
    stopCollectSamples;
end
calibrateGaze;
applyGazeCalibration;
foreach calibration points do
    retrieveTargetStatistics;
end
display calibration results;
```

startCollectSamplesWCS

Description:
Orders SEP to start collecting points for one 3D-point in space given by (x,y,z) position in the World Coordinate System (WCS). The tracked person is assumed to be looking at that target point for the duration of the collection time.

Parameters:

[0]	<int>	Target id. An id to reference the target by when using other calibration requests.
[1]	<double>	The x-component of the 3D-point.
[2]	<double>	The y-component of the 3D-point.
[3]	<double>	The z-component of the 3D-point.
[4]	<int>	Collection timeout in ms.

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "startCollectSamplesWCS",
  "params"  : [2, 0.143, 0.365, 0.512, 2000],
  "id"      : 0
}
```

startCollectSamplesByTargetName

Description:
Orders SEP to start collecting points for a target point with a specific name that is also specified in the loaded World Model. The tracked person is assumed to be looking at that target point for the duration of the collection time.

Parameters:

[0]	<int>	Target id. An id to reference the target by when using other calibration requests.
[1]	<string>	The target name. The World Model loaded in SEP must contain a calibration point with this name.
[4]	<int>	Collection timeout in ms.

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "startCollectSamplesByTargetName",
  "params"  : [2, "calibPoint1", 2000],
  "id"      : 0
}
```

startCollectSamplesObject

Description:
Orders SEP to start collecting points for a target point on the surface of a named object in the World Model. The tracked person is assumed to be looking at that target point for the duration of the collection time.

Parameters:

[0]	<int>	Target id. An id to reference the target by when using other calibration requests.
[1]	<string>	Object name. The object with this name must exist in the World Model loaded in SEP
[1]	<double>	The x-component of the 3D-point in the objects reference system.
[2]	<double>	The y-component of the 3D-point in the objects reference system.
[3]	<double>	The z-component of the 3D-point in the objects reference system.
[3]	<int>	Collection timeout in ms.

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "startCollectSamplesObject",
  "params"  : [2, "LeftScreen", 350, 400, 0, 2000],
  "id"      : 0
}
```

stopCollectSamples

Description:
Stops the samples collection. Useful when you want to stop collecting points before the specified timeout period.

Request Example:

```
{"jsonrpc":"2.0", "method":"stopCollectSamples", "id":0}
```

clearAllTargetSamples

Description:
Clears all the samples for all target points.

Request Example:

```
{"jsonrpc":"2.0", "method":"clearAllTargetSamples", "id":0}
```

clearTargetSamples

Description:
Clears all samples for a single target point.

Parameters:
[0] <int> Target id of the target to be cleared.

Request Example:

```
{"jsonrpc":"2.0", "method":"clearTargetSamples", "params" : [2], "id":0}
```

retrieveTargetStatistics

Description:
Retrieves statistics for a single target. It is assumed that a gaze calibration has been performed on target calibration point before retrieving the data. Left eye is index 0 in the arrays, right eye is index 1.

Parameters:
[0] <int> Target id.

Result:

[targetId]	<int>	Target id
[stdDev]	<double array>	The left and right eye standard deviation of the samples around target calibration point. [Degrees]
[accuracy]	<double array>	The left and right eye accuracy of the calibration around target calibration point. [Degrees]
[errorsxl]	<double array>	The x-component for each sample after calibration relative to the calibration point (left eye). [Degrees]
[errorsyl]	<double array>	The y-component for each sample after calibration relative to the calibration point (left eye). [Degrees]
[errorsxr]	<double array>	The x-component for each sample after calibration relative to the calibration point (right eye). [Degrees]
[errorsyr]	<double array>	The y-component for each sample after calibration relative to the calibration point (right eye). [Degrees]

Request Example:

```
{"jsonrpc":"2.0", "method":"retrieveTargetStatistics", "params" : [2], "id":0}
```

Response Example:

```
{
  "jsonrpc" : "2.0",
  "result"  : {
    "targetId" : 0,
    "stdDev"   : [0.45, 0.462],
    "accuracy" : [0.73, 0.7631],
    "errorsxl" : [-0.0621, 0.3357, 0.2444, 0.1469, ... ],
    "errorsyl" : [0.7448, -0.6037, 0.5173, -0.3936, ... ],
    "errorsxr" : [0.7045, 0.3586, 0.3634, 0.4442, ... ],
    "errorsyr" : [0.7086, 0.8151, -0.1517, -0.5735, ... ]
  },
  "id" : 0
}
```

retrieveCalibrationResults

Description:
Retrieve result from gaze calibration. Left eye is index 0 in the arrays, right eye is index 1.

Result:

[stdDev]	<double array>	The left and right eyes standard deviation of all collected samples for all target points. [Degrees]
[accuracy]	<double array>	The left and right eyes accuracy of all collected samples for all target points. [Degrees]

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "retrieveCalibrationResults",
  "id"      : 0
}
```

Response Example:

```
{"jsonrpc":"2.0","result": {"stdDev": [0.5769, 0.462], "accuracy": [0.8775, 0.7631]}, "id": 0
    }
```

calibrateGaze

Description:
Performs gaze calibration calculations based upon the currently collected samples for all target points. This procedure takes several seconds, so it is recommended to wait for the response of this request in an asynchronous manner. Left eye is index 0 in the arrays, right eye is index 1.

Result:

[stdDev]	<double array>	The left and right eyes standard deviation of all collected samples for all target points. [Degrees]
[accuracy]	<double array>	The left and right eyes accuracy of all collected samples for all target points. [Degrees]

Request Example:

```
{"jsonrpc":"2.0", "method":"calibrateGaze", "id":0}
```

Response Example:

```
{
  "jsonrpc" : "2.0",
  "result"  : {
    "stdDev"   : [0.5769, 0.462],
    "accuracy" : [0.8775, 0.7631]
  },
  "id" : 0
}
```

applyGazeCalibration

Description:
Applies the calculated gaze calibration. The calibration will not be used until this method is called.

Request Example:

```
{"jsonrpc":"2.0", "method":"applyGazeCalibration", "id":0}
```

clearGazeCalibration

Description:
Clears the current gaze calibration.

Request Example:

```
{"jsonrpc":"2.0", "method":"clearGazeCalibration", "id":0}
```

Application control

shutdown

Description:
Shuts SEP down.

Request Example:

```
{"jsonrpc":"2.0", "method":"shutdown", "id":0}
```

keyDown

Description:
Simulates pushing a key on the keyboard. It will continue to be pushed down until the **keyUp** command is called.

Parameters:
[0] <string> The key value.

Request Example:

```
{"jsonrpc":"2.0", "method":"keyDown", "params":["a"], "id":0}
```

keyUp

Description:
Simulates releasing a key on the keyboard.

Parameters:
[0] <string> The key value.

Request Example:

```
{"jsonrpc":"2.0", "method":"keyUp", "params":["a"], "id":0}
```

Notifications

SEP acts as a central notification server. With this functionality you can send generic messages between clients that are connected to SEP.

There is also some built in notifications into SEP that gets sent everytime something specific happens. Clients can register to these notifications and react upon them. A full list of the internal notifications is found in the table below.

Notification name	Description
trackingStarted	Tracking started.
trackingStopped	Tracking stopped.
endOfMovie	Recording playback reached it's end.
collectSamplesStarted	Sample collection started. The same notification is used even if it was started with different kinds of collection methods.
collectSamplesStopped	Sample collection stopped.
calibrationSuccessful	Calibration was successfully calculated.
calibrationFailed	Calibration was unsuccessful.
calibrationApplied	Calibration was applied.
calibrationCleared	Calibration was cleared.
recordingStarted	Recording to file started.
recordingStopped	Recording to file stopped.
recordingError	An error occurred while recording to file. The "params" field of the notification holds additional error information.

subscribeToNotification

Description:
Subscribes to a certain notification message. SEP acts as a notification server.

Parameters:

[0]	<string>	The notification name.

Request Example:

```
{"jsonrpc":"2.0", "method":"subscribeToNotification", "params":["trackingStarted"], "id":0}
```

unsubscribeToNotification

Description:
Unsubscribes to a certain notification message.

Parameters:

[0]	<string>	The notification name.

Request Example:

```
{"jsonrpc":"2.0", "method":"unsubscribeToNotification", "params":["trackingStarted"], "id":0}
```

sendNotification

Description:
Sends a notification message to the server (SEP). The server will then forward the notification message to all clients subscribing to the specific notification name. The forwarded message will be sent as a notification request according to the json-rpc specifications, see 4.1.1.

Parameters:

[0]	<string>	The notification name.
[1]	<string>	[OPTIONAL] Any notification parameters that should be attached to the message. This will be sent as a single string and will not be processed by the server, the receiver is assumed to know how this string should be interpreted.

Request Example:

```
{
  "jsonrpc"  : "2.0",
  "method"   : "sendNotification",
  "params"   : ["stimuliChanged", "computer1;screen2;stim_1.jpg"],
  "id"       : 0
}
```

getCameraImage

Description:
Gets a snapshot from one of the SEP camera(s). The image is serialized down to a base64 encoded string. **This is not efficient and is not meant to be used as a high frame rate video feed.**

Parameters:

[cameraIndex]	<int>	Camera index
[scale]	<double>	Scale of the image (must be 0.01 <= scale <= 10.0)

Result:

[cameraIndex]	<int>	Camera index
[width]	<int>	Image width
[height]	<int>	Image height
[stride]	<int>	The stride. (bytes per line)
[timeStamp]	<string>	The timestamp (see SETimeStamp) of the image
[imageData]	<string>	Raw uint8 image data encoded as base64 string

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "getCameraImage",
  "params"  : [1, 0.01],
  "id"      : 0
}
```

Response Example:

```
{
  "jsonrpc" : "2.0",
  "result"  : {
    "cameraIndex" : 0,
    "width"  : 192,
    "height" : 144,
    "stride" : 192,
    "timeStamp": "405305513",
    "imageData": "CgsLCwsLCgoMCwsLCgoLCws ... LDAoKCgsKCgsLCwoLCgsK"
  },
  "id" : 0
}
```

isCameraConnectedToUSB3

Description:
Check if a SEP camera(s) is connected to a USB3 port.

Parameters:

[cameraIndex]	<int>	Camera index.

Result:

[connectedToUSB3]	<bool>	Is camera connected to USB3 or not.

Request Example:

```
{
  "jsonrpc" : "2.0",
  "method"  : "isCameraConnectedToUSB3",
  "params"  : [0],
  "id"      : 0
}
```

Response Example:

```
{
  "jsonrpc" : "2.0",
  "result"  : {
```

```
    "connectedToUSB3" : True
  },
  "id" : 0
}
```

4.2 Exit Codes

The following exit codes may be returned by SEP:

Code	Details
0	Normal exit
1000	Another instance of SEP is already running.
2000	Smart Eye Tracker start flag is missing. *(Smart Eye Tracker (SET) only)*
3000	Camera is not connected. *(SET only)*
4000	Camera is not connected to a USB3 port. *(SET only)*
5000	Camera is already in use. *(SET only)*
6000	Invalid command line parameters.

Chapter 5

Time Synchronization Specification

The Time Synchronization Module has two main features:

- It enables the user to provide a user defined 64 bits value as a timestamp to each frame by writing a very simple dll. This is called from the *Smart Eye* system at the moment of exposure for each frame. The UserTimestamp is passed on to all logs and makes it easy to synchronize data from the *Smart Eye* system with data from another system utilizing the same timestamp. Instructions on writing the dll follow below.
- The Time Synchronization module also provides information about latency for each frame, which is the time from the image capture to the time the log data leaves the application either on file, TCP, UDP or rs232. This is important in real time applications. Note that this value differs by several 100 microseconds for different data streams, since they are not send at the exact same time.

5.1 Writing a User Time dll

The installation provides an example on how to implement a dll, see `API\examples\HighResolutionTime` of the SEP installation directory. The header file specifying the function implemented by the dll is named `TimeService.h` and found in `API\include` in the installation directory.

Compiling the HighResolutionTime project results in the default HighResolutionTime.dll, which provides the realtime clock of the PC.

The dll has four functions that may be modified by the user. The `getCurrentTime` function which should return a 64-bit value representing the UserTimestamp. If necessary, start-up and close-down code can be added into the `startTimeService` and `stopTimeService` functions respectively. The `timeToString` should be able to convert the UserTimestamp to a readable value that may be presented in the user interface.

The `getCurrentTime` function of the dll is called upon capture of a video frame from the cameras. The capture is halted while waiting for a response, thus the `getCurrentTime` function must return very fast and must be non-blocking. Ideally only calls to Windows system dlls should be performed and no locks or any other pieces of code that may halt the execution can be utilized.

Also, the `getCurrentTime` function should ideally be reentrant, meaning that more than one thread can run the function simultaneously without hazards. By default the system expects that the function is reentrant, but this may be configured in the `Settings`→`Software...`→`User Time Stamp`→`Current Timestamp DLL` tab.

The sample code also contains a project called "HighResolutionTimeTest", which is an application for testing the HighResolutionTime.dll outside SEP. The HighResolutionTime.exe is run om two clients (can be on the same computer or two different), and the usage is:

```
TimeServiceTest.exe <DLLFileName> <offset> <remoteHost> <clientType>
```

- `DLLFileName` is the full path to the dll.

- `offset` is an optional time to add to the current time, set this to 0
- `remoteHost` is the ip address of the other client
- `clientType` is either a or b, one client is called a and the other b.

At run time SEP will add an offset to the time elapsed from the midpoint of camera exposure until the time when `getCurrentTime` is called.

5.1.1 startTimeService

Start and possibly synchronize the time service, allocate any resources needed. This function is called exactly once (per client) before any other calls are made from that client.

```
TIMESERVICE_API( startTimeService) ( TTimeServiceHandle * handle );
```

Parameter	I/0	Description
handle	0	Pointer to a time service handle

5.1.2 stopTimeService

Stop time service and release resources. This function is called exactly once (per client) after all other calls have been made.

```
TIMESERVICE_API( stopTimeService ) ( TTimeServiceHandle handle );
```

Parameter	I/0	Description
handle	I	time service handle

5.1.3 getCurrentTime

Get the current time in user format. A time offset should be added to the result in order to get the actual time of a past or future event. This function can be called from time-critical sections of the code and should avoid potentially blocking calls and return as soon as possible.

```
TIMESERVICE_API( getCurrentTime ) (
  TTimeServiceHandle handle,
  signed __int64 offset,
  TOpaqueUserTime * currentTime);
```

Parameter	I/0	Description
handle	I	time service handle
offset	I	offset in 100 ns units
currentTime	0	current time in user format, adjusted by offset

5.1.4 timeToString

Convert a user format timestamp to a human readable text format. The text format is intended for presentation and not data logging, so a one-to-one mapping is not required.

```
TIMESERVICE_API(timeToString) (
  TTimeServiceHandle handle,
  TOpaqueUserTime currentTime,
  unsigned long * bufferSize,
  char * textBuffer);
```

Parameter	I/0	Description
handle	I	time service handle
currentTime	0	current time in user format, adjusted by offset
bufferSize	I/0	size of text buffer in bytes incl null termination (in = max, out = actual)
textBuffer	I	time in text format (null terminated)

Chapter 6

Example Applications

6.1 Socket Client

The SocketClient example implements an application in C++ for connecting to and receiving output data from SEP via a TCP or UDP socket.

The SocketClient example files are located in the folder `API\Examples\SocketClient` in the installation path of SEP. Please refer to the the README.md file in that folder for further documentation of the SocketClient example project.

6.2 Can Client

All example files are located in the folder `API\Examples\CanClient` in the installation path of SEP.

This client is an example of how to connect using the CANcaseXL box and receiving messages. Parts of this code can be copied into your own client or used this code to build your client from. The program has an infinite loop that

1. Wait for message
2. Check message type
3. Interpret contents of packet(print content and call user function)
4. Return waiting for packets

6.2.1 receiveMessage

This function takes an event from the CANCaseXL box, determines message type and interpret the contents.

```
int receiveMessage( XLevent xlEvent );
```

Parameter	I/0	Description
xlEvent	I	A struct containing an event from the CANCaseXL box

6.2.2 receive<messageType>Message

This function parses the data from a message and prints the contents to standard out. Valid message types are

- Sync
- UserTimeStamp
- HeadPosition
- HeadRotation
- GazeOrigin
- GazeDirection
- EyeClosure
- BothEyeClosure
- PupilDiameter
- BothPupilDiameter

```
int receive<messageType>Message(unsigned char* message);
```

Parameter	I/0	Description
message	I	Pointer to the beginning of the message data buffer

6.2.3 initDriver

A function that initiates the connection to the CANCaseXL box.

```
XLstatus initDriver( CANBusInfo* canInfo, char* appName );
```

Parameter	I/0	Description
canInfo	I	A struct containing information about the CAN connection
appName	I	Name of the application in the Vector Hardware configuration

6.3 TimeService

All example files are located in the folder `API\examples\HighResolutionTime` in the installation path of SEP. Read Chapter 5 for more information.

6.4 Python Examples

There is a suite of Python scripts available for usage together with the SEP application. Below is a list of the available files with a short explanation. The scripts and usage of them are described in details in separate subsections further below. All scripts require that Python 3 is installed. The scripts can be found in chosen installation folder under API\Examples\PythonExamples.

Script	Description
can_types.py	Module containing type structure definitions used by can_client.py.
can_client.py	Module containing functionality regarding receiving output data on CAN bus.
json_log_can	File containing JSON structured CAN output data.
socket_types.py	Module containing type structure definitions used by socket_client.py.
socket_client.py	Module containing functionality regarding receiving output data on TCP/UDP socket
json_log_socket	File containing JSON structured socket output data.
client.py	Contains examples on how to use can_client.py and socket_client.py.
external_interfaces.py	Module containing functionality regarding communication with SEP over RPC interface
command_line.py	Command line interface wrapping functionality in external_interfaces.py.
log_data_list.txt	Contains output message names that can be requested over socket connection.

6.4.1 can_client.py

The can_client module is intended to help setting up a CAN bus connection and start receiving output data. The module can be imported into other scripts and be used according to example below.

Prerequisite:
CAN output enabled in SEP application according to SEP manual.
CAN hardware or virtual CAN bus set up to receive CAN output data according to SEP manual.
Python 3 version installed.
Python package pywin32 installed.

Example on how to use can_client module:

```
import can_client
client = can_client.CanClient()

client.init()
client.receive()
```

6.4.2 socket_client.py

The socket_client module is intended to help setting up a TCP/UDP connection and start receiving output data. The module can be imported into other scripts and be used according to example below.

Prerequisite:
TCP/UDP output enabled in SEP application according to SEP manual.

Example on how to use socket_client module:

```
import socket_client
client = socket_client.SocketClient()

client.init("127.0.0.1", 5002, "tcp")
client.connect()
client.receive()
```

6.4.3 client.py

This module contains usage examples of socket_client and can_client modules. It includes the option to log output data between a range of frames along with examples on how to parse output logs json_log_socket and json_log_can.

Prerequisite:
Same as socket_client and can_client modules depending on which mode is specified.

Example on how to use client module:

1. Open the windows command line.
2. Navigate to the folder where the script resides.
3. Run the following to see the options available:

   ```
   py -3 client.py -h
   ```

4. Specify command line options and run script again. Example:

   ```
   python -3 client.py --mode can
   ```

6.4.4 external_interfaces.py

The external_interfaces module contains functionality needed to communicate with the SEP application over RPC interface. It is intended to simplify the process of writing scripts to interact with the SEP application. Below is a simple example of how to connect and send a ping to the SEP application. Other commands can be added in the same fashion.

Prerequisite:
SEP application running.

Example on how to use external_interfaces module:

```
import external_interfaces

IP = "127.0.0.1"
PORT = "8100"

ext_interface = external_interfaces.ExternalInterface()
ext_interface.connect(IP, PORT)
ext_interface.send_ping()
```

6.4.5 command_line.py

The command_line module provides a command line interface that can be used to communicate with SEP application over RPC interface. It wraps the functionality of external_interfaces.py and is intended to be used as is. It is also a good reference example on how to use the external_interfaces module.

Prerequisite:
SEP application running.

Example on how to use command_line module:

1. Open the windows command line.
2. Navigate to the folder where the script resides.
3. Run from command line with:

   ```
   py -3 command_line.py
   ```

6.5 Broadcast Example

All example files are located in the folder `API\Examples\BroadcastExample` in the installation path of SEP. Open the project file in Visual C++ and build the executable to test the example. A prebulit executable is included in the folder `API\Examples\bin` in the installation path of SEP.

This is an example of how to receive broadcasted video feeds from multiple cameras. The program will look up all the Basler GigE cameras connected to the system and go through the necessary steps for receiving a video feed from each camera. The images received will be displayed in a separate window for each camera for a specified number of frames. The port used defaults to the same port value as SEP but can be specified for each camera.

6.6 RPC Example

The RPCExample shows how to use the SeJsonRPC.dll (see section 4.1.5 Example DLL) to communicate with Smart Eye Pro from a C++ application. Please refer to the README.md file in the `API\Examples\RPCExample` folder for setup and build instructions.

Chapter 7

Scripts

7.1 Matlab

7.1.1 Log2Matlab

Log2Matlab is an application that converts *Smart Eye* log-files into *Matlab*®'s binary data-files (.mat). Multiple log files can be entered and the generated mat-files get the same names as the respective log file. Each column in the log file is stored as a matrix with the same name as the column. If the column is a 3D point, 3D vector or an other type with multiple values the matrix has multiple rows. Finally the six rows in an intersection is stored in `worldPoint.x-z` followed by `objectPoint.x-z`.

> **Note**
>
> *If the* Matlab® *-file is going to be used in* SciLab *it is important to set* Scilab *flag in the settings. This will shorten variable names that are too long since this is not supported by* Scilab.

7.1.2 Saccade and Fixation Analysis

All *Matlab*® files are located in the folder `API\utilities\Matlab` in the installation path of SEP.

saccadeAndFixationAnalysis

The file saccadeAndFixationAnalysis.m contains a *Matlab*® function that analyses the gaze data for saccades and fixations. It draws graphs to show the result of the analysis.

```
SaccadeAndFixationAnalysis( leftGazeDirection, leftGazeDirectionQ, rightGazeDirection,
rightGazeDirectionQ, timeStamps )
```

Parameter	I/O	Description
leftGazeDirection	I	The leftGazeDirection column from a log.
leftGazeDirectionQ	I	The leftGazeDirectionQ column from a log.
rightGazeDirection	I	The rightGazeDirection column from a log.
rightGazeDirectionQ	I	The rightGazeDirectionQ column from a log.
timeStamps	I	The timestamp column from a log.

toPolarDeg

A *Matlab*®function that convert 3D gaze direction vector to polar coordinates in degrees.

```
polarGaze = toPolarDeg(aGazeDirection)
```

Parameter	I/O	Description
aGazeDirection	I	A 3D gaze direction vector.
polarGaze	0	The 3D gaze direction vector in polar coordinates in degrees.

removeLowQ

A *Matlab*®function that set gaze direction to NaN where the quality of the gaze is less than a threshold.

```
prunedGazeDirection = removeLowQ( gazeDirection, gazeDirectionQ, threshold )
```

Parameter	I/O	Description
gazeDirection	I	A vector of gaze directions.
gazeDirectionQ	I	A vector of gaze direction qualities.
threshold	I	The threshold for the qualities.
prunedGazeDirection	0	A vector of accepted gaze directions and/or NaN values.

medianFilt

A *Matlab*®function that applies a median filter of size length and ignores NaN values that exist in the input data.

```
vecOut = medianFilt(vecIn, length)
```

Parameter	I/O	Description
vecIn	I	A vector of input data.
length	I	Size of filter.
vecOut	0	A vector of filtered values.

rotationVelocity

A *Matlab*®function that calculates the horizontal (H), vertical(V) and diagonal ($D = \sqrt{H^2 + V^2}$) rotational velocity of a set of gaze directions.

```
vel = rotationVelocity(gazeDirection,timeSec)
```

Parameter	I/O	Description
gazeDirection	I	A vector of gaze directions.
timeSec	I	A vector times in seconds.
vel	I	A vector of rotational velocities.

7.2 Scilab

All *Matlab*®files are located in the folder `API\utilities\Scilab` in the installation path of SEP.

7.2.1 Importing Logfiles

The file getData.sci contains functions to import data from *Smart Eye* log-files into the program *SciLab*. It is also possible with the module *Log2Matlab* (described in section 7.1.1) to import logs into *Scilab*.

getDataFromFile

Used to load data from a *Smart Eye* log-file into a matrix.

```
[data, headers] = getDataFromFile(filename)
```

Parameter	I/O	Description
filename	I	Tab separated smart eye log file.
data	0	Matrix of log data.
headers	0	Vector of column names.

getData

Used to load data from a *Smart Eye* log-file, opens a dialog for choosing which log file to open.

```
[data, headers] = getData()
```

Parameter	I/O	Description
data	0	Matrix of log data.
headers	0	Vector of column names.

getColumnByName

Used to get the column vector for the specified name of a column in the log-file, for example `GazeDirectionQ` or `GazeDirection.x`.

```
[col] = getColumnByName(data,header,name)
```

Parameter	I/O	Description
data	0	Matrix of *Smart Eye Pro* data.
headers	0	Vector of column names.
name	I	The name of the desired log data.
col	0	A vector of the desired data.

getXYZColumnsByName

Used to get the column vectors, one for each component of a 3-dimensional data type, for the specified name, for example `GazeDirection`.

```
[cols] = getXYZColumnsByName(data,header,name)
```

Parameter	I/O	**Description**
data	0	Matrix of *Smart Eye Pro* data.
headers	0	Vector of column names.
name	I	The name of the desired log data.
col	0	A matrix (3 cols) of the desired data.

Appendix A

Smart Eye output data ids

Table A.1: A List of all the available Smart Eye Data, their ID and Data Type.

DataID	**EnumValue**	**DataType**
SEFrameNumber	0x0001	SEType_u32
SEEstimatedDelay	0x0002	SEType_u32
SETimeStamp	0x0003	SEType_u64
SEUserTimeStamp	0x0004	SEType_u64
SEFrameRate	0x0005	SEType_float
SECameraPositions	0x0006	SEType_Vector<SEType_Point3D>
SECameraRotations	0x0007	SEType_Vector<SEType_Vect3D>
SEUserDefinedData	0x0008	SEType_u64
SERealTimeClock	0x0009	SEType_u64
SEHeadPosition	0x0010	SEType_Point3D
SEHeadPositionQ	0x0011	SEType_float
SEHeadRotationRodrigues	0x0012	SEType_Vect3D
SEHeadRotationQuaternion	0x001d	SEType_Quaternion
SEHeadLeftEarDirection	0x0015	SEType_Vect3D
SEHeadUpDirection	0x0014	SEType_Vect3D
SEHeadNoseDirection	0x0013	SEType_Vect3D
SEHeadHeading	0x0016	SEType_float
SEHeadPitch	0x0017	SEType_float
SEHeadRoll	0x0018	SEType_float
SEHeadRotationQ	0x0019	SEType_float
SEGazeOrigin	0x001a	SEType_Point3D
SELeftGazeOrigin	0x001b	SEType_Point3D
SERightGazeOrigin	0x001c	SEType_Point3D
SEEyePosition	0x0020	SEType_Point3D
SEGazeDirection	0x0021	SEType_Vect3D
SEGazeDirectionQ	0x0022	SEType_float
SELeftEyePosition	0x0023	SEType_Point3D
SELeftGazeDirection	0x0024	SEType_Vect3D
SELeftGazeDirectionQ	0x0025	SEType_float
SERightEyePosition	0x0026	SEType_Point3D
SERightGazeDirection	0x0027	SEType_Vect3D
SERightGazeDirectionQ	0x0028	SEType_float
SEGazeHeading	0x0029	SEType_float
SEGazePitch	0x002a	SEType_float
SELeftGazeHeading	0x002b	SEType_float

Table A.1: A List of all the available Smart Eye Data, their ID and Data Type.

DataID	EnumValue	DataType
SELeftGazePitch	0x002c	SEType_float
SERightGazeHeading	0x002d	SEType_float
SERightGazePitch	0x002e	SEType_float
SEFilteredGazeDirection	0x0030	SEType_Vect3D
SEFilteredLeftGazeDirection	0x0032	SEType_Vect3D
SEFilteredRightGazeDirection	0x0034	SEType_Vect3D
SEFilteredGazeHeading	0x0036	SEType_float
SEFilteredGazePitch	0x0037	SEType_float
SEFilteredLeftGazeHeading	0x0038	SEType_float
SEFilteredLeftGazePitch	0x0039	SEType_float
SEFilteredRightGazeHeading	0x003a	SEType_float
SEFilteredRightGazePitch	0x003b	SEType_float
SESaccade	0x003d	SEType_u32
SEFixation	0x003e	SEType_u32
SEBlink	0x003f	SEType_u32
SEClosestWorldIntersection	0x0040	SEType_WorldIntersection
SEFilteredClosestWorldIntersection	0x0041	SEType_WorldIntersection
SEAllWorldIntersections	0x0042	SEType_WorldIntersections
SEFilteredAllWorldIntersections	0x0043	SEType_WorldIntersections
SEZoneId	0x0044	SEType_u16
SEEstimatedClosestWorldIntersection	0x0045	SEType_WorldIntersection
SEEstimatedAllWorldIntersections	0x0046	SEType_WorldIntersections
SEHeadClosestWorldIntersection	0x0049	SEType_WorldIntersection
SEHeadAllWorldIntersections	0x004a	SEType_WorldIntersections
SEEyelidOpening	0x0050	SEType_float
SEEyelidOpeningQ	0x0051	SEType_float
SELeftEyelidOpening	0x0052	SEType_float
SELeftEyelidOpeningQ	0x0053	SEType_float
SERightEyelidOpening	0x0054	SEType_float
SERightEyelidOpeningQ	0x0055	SEType_float
SEKeyboardState	0x0056	SEType_String
SELeftLowerEyelidExtremePoint	0x0058	SEType_Point3D
SELeftUpperEyelidExtremePoint	0x0059	SEType_Point3D
SERightLowerEyelidExtremePoint	0x005a	SEType_Point3D
SERightUpperEyelidExtremePoint	0x005b	SEType_Point3D
SEPupilDiameter	0x0060	SEType_float
SEPupilDiameterQ	0x0061	SEType_float
SELeftPupilDiameter	0x0062	SEType_float
SELeftPupilDiameterQ	0x0063	SEType_float
SERightPupilDiameter	0x0064	SEType_float
SERightPupilDiameterQ	0x0065	SEType_float
SEFilteredPupilDiameter	0x0066	SEType_float
SEFilteredPupilDiameterQ	0x0067	SEType_float
SEFilteredLeftPupilDiameter	0x0068	SEType_float
SEFilteredLeftPupilDiameterQ	0x0069	SEType_float
SEFilteredRightPupilDiameter	0x006a	SEType_float
SEFilteredRightPupilDiameterQ	0x006b	SEType_float
SEGPSPosition	0x0070	SEType_Point2D
SEGPSGroundSpeed	0x0071	SEType_float
SEGPSCourse	0x0072	SEType_float

Table A.1: A List of all the available Smart Eye Data, their ID and Data Type.

DataID	EnumValue	DataType
SEGPSTime	0x0073	SEType_u64
SEEstimatedGazeOrigin	0x007a	SEType_Point3D
SEEstimatedLeftGazeOrigin	0x007b	SEType_Point3D
SEEstimatedRightGazeOrigin	0x007c	SEType_Point3D
SEEstimatedEyePosition	0x0080	SEType_Point3D
SEEstimatedGazeDirection	0x0081	SEType_Vect3D
SEEstimatedGazeDirectionQ	0x0082	SEType_float
SEEstimatedGazeHeading	0x0083	SEType_float
SEEstimatedGazePitch	0x0084	SEType_float
SEEstimatedLeftEyePosition	0x0085	SEType_Point3D
SEEstimatedLeftGazeDirection	0x0086	SEType_Vect3D
SEEstimatedLeftGazeDirectionQ	0x0087	SEType_float
SEEstimatedLeftGazeHeading	0x0088	SEType_float
SEEstimatedLeftGazePitch	0x0089	SEType_float
SEEstimatedRightEyePosition	0x008a	SEType_Point3D
SEEstimatedRightGazeDirection	0x008b	SEType_Vect3D
SEEstimatedRightGazeDirectionQ	0x008c	SEType_float
SEEstimatedRightGazeHeading	0x008d	SEType_float
SEEstimatedRightGazePitch	0x008e	SEType_float
SEFilteredEstimatedGazeDirection	0x0091	SEType_Vect3D
SEFilteredEstimatedGazeDirectionQ	0x0092	SEType_float
SEFilteredEstimatedGazeHeading	0x0093	SEType_float
SEFilteredEstimatedGazePitch	0x0094	SEType_float
SEFilteredEstimatedLeftGazeDirection	0x0096	SEType_Vect3D
SEFilteredEstimatedLeftGazeDirectionQ	0x0097	SEType_float
SEFilteredEstimatedLeftGazeHeading	0x0098	SEType_float
SEFilteredEstimatedLeftGazePitch	0x0099	SEType_float
SEFilteredEstimatedRightGazeDirection	0x009b	SEType_Vect3D
SEFilteredEstimatedRightGazeDirectionQ	0x009c	SEType_float
SEFilteredEstimatedRightGazeHeading	0x009d	SEType_float
SEFilteredEstimatedRightGazePitch	0x009e	SEType_float
SEASCIIKeyboardState	0x00a4	SEType_u16
SECalibrationGazeIntersection	0x00b0	SEType_WorldIntersection
SETaggedGazeIntersection	0x00b1	SEType_WorldIntersection
SELeftClosestWorldIntersection	0x00b2	SEType_WorldIntersection
SELeftAllWorldIntersections	0x00b3	SEType_WorldIntersections
SERightClosestWorldIntersection	0x00b4	SEType_WorldIntersection
SERightAllWorldIntersections	0x00b5	SEType_WorldIntersections
SEFilteredLeftClosestWorldIntersection	0x00b6	SEType_WorldIntersection
SEFilteredLeftAllWorldIntersections	0x00b7	SEType_WorldIntersections
SEFilteredRightClosestWorldIntersection	0x00b8	SEType_WorldIntersection
SEFilteredRightAllWorldIntersections	0x00b9	SEType_WorldIntersections
SEEstimatedLeftClosestWorldIntersection	0x00ba	SEType_WorldIntersection
SEEstimatedLeftAllWorldIntersections	0x00bb	SEType_WorldIntersections
SEEstimatedRightClosestWorldIntersection	0x00bc	SEType_WorldIntersection
SEEstimatedRightAllWorldIntersections	0x00bd	SEType_WorldIntersections
SELeftBlinkClosingMidTime	0x00e0	SEType_u64
SELeftBlinkOpeningMidTime	0x00e1	SEType_u64
SELeftBlinkClosingAmplitude	0x00e2	SEType_float
SELeftBlinkOpeningAmplitude	0x00e3	SEType_float

Table A.1: A List of all the available Smart Eye Data, their ID and Data Type.

DataID	**EnumValue**	**DataType**
SELeftBlinkClosingSpeed	0x00e4	SEType_float
SELeftBlinkOpeningSpeed	0x00e5	SEType_float
SERightBlinkClosingMidTime	0x00e6	SEType_u64
SERightBlinkOpeningMidTime	0x00e7	SEType_u64
SERightBlinkClosingAmplitude	0x00e8	SEType_float
SERightBlinkOpeningAmplitude	0x00e9	SEType_float
SERightBlinkClosingSpeed	0x00ea	SEType_float
SERightBlinkOpeningSpeed	0x00eb	SEType_float
SELeftEyelidState	0x0390	SEType_u8
SERightEyelidState	0x0391	SEType_u8
SEUserMarker	0x03a0	SEType_UserMarker
SECameraClocks	0x03a1	SEType_Vector<SEType_u64>

Appendix B

Smart Eye output data descriptions

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
FrameNumber		A sequential frame number, set by the image capture subsystem. The frame number starts counting from 0 at application start and increments by +1 for each successfully captured image. This means that the frame number of two consecutive frames will always differ by 1 even if the image capture subsystem should happen to drop one or more frames in between. If the frame numbers in the data output should increment by more than +1 from one data record to the next, then it is either an indication of frame loss in the image processing subsystem, probably due to CPU overload or, in the case of UDP communication, loss of network data packets. If an image source error, such as an Ethernet cable temporarily loosing connection to its socket, occurs when Smart Eye Pro is running the frame numbering will restart from 0 to indicate that a problem has occurred. N.B: The frame number should not be used to detect frame loss. The only reliable way to determine frame loss is to check the time stamp difference between two consecutive data records.
EstimatedDelay	100ns	The estimated delay from the real-world event (midpoint of image exposure) to the network socket send() system call. **[Time Sync]**
TimeStamp	100ns	A high-resolution time stamp, measured at the midpoint of image exposure and starting to count from 0 when the application is started. Since this time stamp is based on the PC hardware high-resolution clock, it should only be used to measure time differences over relatively short periods of time.

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
UserTimeStamp	User defined	64 bits of user defined information, usually an absolute-time time stamp and/or annotation data. The .dll generating this value has to follow the TimeService.h interface (See Chapter 5 for detailed information). **[Time Sync]**
FrameRate	Hz	The current frame rate of the system in frames. Should ideally be 60 or 120 frames per second depending on system configuration.
CameraPositions	m	The position of all the cameras defined in the World Coordinate System.
CameraRotations	rad	The rodrigues rotation of all the cameras defined in the World Coordinate System.
UserDefinedData		User defined data, not used at the moment. **[UserDefinedData]**
RealTimeClock	100ns	The real time clock reads the computer clock and represents it as a 64-bit value in FILETIME format as an absolute time since January 1, 1601 (UTC).
HeadPosition	m	The 3D head position in the defined World Coordinate System.
HeadPositionQ		Quality of the head position [0..1] (0.0 = no head tracking, 0.05..0.1 = Face Detection, 0.2 = Face Refinder, 0.2..1.0 = Head Tracking).
HeadRotationRodrigues	rad	The 3D head orientation in Rodrigues format in the defined World Coordinate System.
HeadRotationQuaternion		The 3D head orientation as a quaternion in the defined World Coordinate System.
HeadLeftEarDirection		Unit vector defining the 'left ear' direction. Identical to the x-axis of the head rotation matrix.
HeadUpDirection		Unit vector defining the 'up' direction. Identical to the y-axis of the head rotation matrix.
HeadNoseDirection		Unit vector defining the 'nose' direction. Identical to the z-axis of the head rotation matrix.
HeadHeading	rad	The left/right-rotation of the head. Also known as "no"-rotation (See Section on "Definition of Euler Angles" in the SEP manual).
HeadPitch	rad	The up/down-rotation of the head. Also known as "yes"-rotation (See Section on "Definition of Euler Angles" in the SEP manual).
HeadRoll	rad	The tilt-rotation of the head. Also known as "maybe"-rotation (See Section on "Definition of Euler Angles" in the SEP manual).
HeadRotationQ		Quality of the head orientation [0..1] (The same value as the head position quality).
GazeOrigin	m	The consensus of the LeftGazeOrigin and RightGazeOrigin.
LeftGazeOrigin	m	The center of the pupil/iris of the left eye, where the gaze vector originates.
RightGazeOrigin	m	The center of the pupil/iris of the right eye, where the gaze vector originates.

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
EyePosition	m	The virtual 3D eye position given in the defined World Coordinate System.
GazeDirection		A unit vector originating in the virtual eye position describing the direction of the gaze. Calculated as the average of the two gaze vectors originating at the physical eyes. If vergence is of interest, the eyes should be handled separately instead.
GazeDirectionQ		Quality of the gaze direction measurement [0..1]. Depends on the quality of the pupil, iris and glint detection and the degree of agreement between measurements from different eye clips. Calculated as the maximum of the quality of the both physical eyes (see Section 3.6.2).
LeftEyePosition	m	The 3D position of the center of the left eye ball given in the defined World Coordinate System.
LeftGazeDirection		A unit vector describing the direction of the gaze of the left eye originating from the LeftGazeOrigin.
LeftGazeDirectionQ		Quality of the left gaze direction measurement [0..1]. Depends on the quality of the pupil, iris and glint detection of the left eye and the degree of agreement between measurements from different eye clips for the left eye (see Section 3.6.2).
RightEyePosition	m	The 3D position of the center of the right eye ball given in the defined World Coordinate System.
RightGazeDirection		A unit vector describing the direction of the gaze of the right eye, originating from the LeftGazeOrigin.
RightGazeDirectionQ		Quality of the right gaze direction measurement [0..1]. Depends on the quality of the pupil, iris and glint detection of the right eye and the degree of agreement between measurements from different eye clips for the right eye (see Section 3.6.2).
GazeHeading	rad	The left/right angle of the GazeDirection (See Section on "Definition of Euler Angles" in the SEP manual).
GazePitch	rad	The up/down angle of the GazeDirection (See Section on "Definition of Euler Angles" in the SEP manual).
LeftGazeHeading	rad	The left/right angle of the LeftGazeDirection (See Section on "Definition of Euler Angles" in the SEP manual).
LeftGazePitch	rad	The up/down angle of the LeftGazeDirection (See Section on "Definition of Euler Angles" in the SEP manual).
RightGazeHeading	rad	The left/right angle of the RightGazeDirection (See Section on "Definition of Euler Angles" in the SEP manual).

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
RightGazePitch	rad	The up/down angle of the RightGazeDirection (See Section on "Definition of Euler Angles" in the SEP manual).
Saccade		When the saccade detection filter detects a saccade, this value will be a non-zero integer for the duration of the saccade. This integer increases for each saccade and can be seen as a unique saccade identifier.
Fixation		When the fixation detection filter detects a fixation, this value will be a non-zero integer for the duration of the fixation. This integer increases for each fixation and can be seen as a unique fixation identifier.
Blink		When the blink detection filter detects a blink, this value will be a non-zero integer for the duration of the blink. This integer increases for each blink and can be seen as a unique blink identifier.
ClosestWorldIntersection		The closest gaze intersection with any of the world objects. The intersection information contains name of object, intersection point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame this integer will be 0, otherwise 1.
AllWorldIntersections		Analogue to ClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected. As with ClosestWorldIntersections, each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. The difference is that in this case there may be any number of intersections.
FilteredAllWorldIntersections		Analogue to FilteredClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected.As with ClosestWorldIntersections, each sub packet of this type starts with an integer indi- cating the number of world intersections con- tained in the sub packet. The difference is that in this case there may be any number of intersections.
NumericalClosestWorldIntersection		Obsolete. Will be removed.

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
EstimatedClosestWorldIntersection		The closest intersection of the estimated gaze vector with any of the world objects. The intersection information contains name of object, intersection point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame this integer will be 0, otherwise 1.
EstimatedAllWorldIntersections		Analogue to EstimatedClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected. As with ClosestWorldIntersections, each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. The difference is that in this case there may be any number of intersections.
HeadClosestWorldIntersection		The closest intersection of the head nose vector with any of the world objects. The intersection information contains name of object, intersection point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame this integer will be 0, otherwise 1.
HeadAllWorldIntersections		Analogue to HeadClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected. As with ClosestWorldIntersections, each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. The difference is that in this case there may be any number of intersections.
EyelidOpening	m	The average distance between the eyelids of both eyes.
EyelidOpeningQ		Normally in the range 0..1, some subjects may have values larger than 1.0. The value depends on how distinct the eyelids can be detected. The range is individual and can not be compared between subjects. Calculates as the average quality of the both physical eyes.
LeftEyelidOpening	m	The distance between the eyelids of the left eye.

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
LeftEyelidOpeningQ		Quality of left eyelid detection (See description for EyelidOpeningQ).
RightEyelidOpening	m	The distance between the eyelids of the right eye.
RightEyelidOpeningQ		Quality of right eyelid detection (See description for EyelidOpeningQ).
KeyboardState		The keys that are currently pressed (A-Z, 0-9), useful for marking events in log files or movie recordings. Please note that due to limitations in a normal PC-keyboard, pressing more than two keys at a time gives undefined output.
LeftLowerEyelidExtremePoint	m	The midpoint of the eyelid represented in the 3D coordinates of the World Coordinate System.
LeftUpperEyelidExtremePoint	m	The midpoint of the eyelid represented in the 3D coordinates of the World Coordinate System.
RightLowerEyelidExtremePoint	m	The midpoint of the eyelid represented in the 3D coordinates of the World Coordinate System.
RightUpperEyelidExtremePoint	m	The midpoint of the eyelid represented in the 3D coordinates of the World Coordinate System.
PupilDiameter	m	The consensus diameter of the pupils of both eyes. **[Pupilometry]**
PupilDiameterQ		The quality value for the PupilDiameter. The value depends on how distinct the pupil/iris edge can be detected. The value is normalized and will be in the range of [0.0, 1.0]. **[Pupilometry]**
LeftPupilDiameter	m	The diameter of the pupil of the left eye. **[Pupilometry]**
LeftPupilDiameterQ		The quality value for the LeftPupilDiameter. The value depends on how distinct the pupil/iris edge can be detected. The value is normalized and will be in the range of [0.0, 1.0]. **[Pupilometry]**
RightPupilDiameter	m	The diameter of the pupil of the right eye. **[Pupilometry]**
RightPupilDiameterQ		The quality value for the RightPupilDiameter. The value depends on how distinct the pupil/iris edge can be detected. The value is normalized and will be in the range of [0.0, 1.0]. **[Pupilometry]**
EstimatedGazeOrigin	m	The calculation of the estimated gaze is based on the estimated eye center as obtained from head tracking. Its origin is the pupil or iris.

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
EstimatedLeftGazeOrigin	m	The calculation of the estimated gaze is based on the estimated eye center as obtained from head tracking. Its origin is the pupil or iris.
EstimatedRightGazeOrigin	m	The calculation of the estimated gaze is based on the estimated eye center as obtained from head tracking. Its origin is the pupil or iris.
EstimatedEyePosition	m	The position of the consensus eye as obtained from head tracking.
EstimatedGazeDirection		The calculation of the estimated gaze is based on the estimated eye center as obtained from head tracking. This is the direction is the vector from estimated eye center to pupil or iris.
EstimatedGazeDirectionQ		The calculation of the estimated gaze is based on the estimated eye center as obtained from head tracking. The gaze direction quality depends on how distinct the pupil/iris can be detected and how well gaze measurements from different eye clips coincide.
EstimatedGazeHeading	rad	The heading angle of the estimated consensus gaze direction. (See Section on "Definition of Euler Angles" in the SEP manual).
EstimatedGazePitch	rad	The pitch angle of the estimated consensus gaze direction. (See Section on "Definition of Euler Angles" in the SEP manual).
EstimatedLeftEyePosition	m	The position of the left eye as obtained from head tracking.
EstimatedLeftGazeDirection		The calculation of the estimated gaze is based on the estimated eye center as obtained from head tracking. This is the normalized vector from the left estimated eye center to pupil or iris.
EstimatedLeftGazeDirectionQ		The calculation of the estimated gaze is based on the estimated eye center as obtained from head tracking. The gaze direction quality depends on how distinct the pupil/iris can be detected and how well gaze measurements from several left eye clips coincide.
EstimatedLeftGazeHeading	rad	The heading angle of the estimated left gaze direction. (See Section on "Definition of Euler Angles" in the SEP manual).
EstimatedLeftGazePitch	rad	The pitch angle of the estimated left gaze direction. (See Section on "Definition of Euler Angles" in the SEP manual).
EstimatedRightEyePosition	m	The position of the right eye as obtained from head tracking.
EstimatedRightGazeDirection		The calculation of the estimated gaze is based on the estimated eye center as obtained from head tracking. This is the normalized vector from the right estimated eye center to pupil or iris.

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
EstimatedRightGazeDirectionQ		The calculation of the estimated gaze is based on the estimated eye center as obtained from head tracking. The gaze direction quality depends on how distinct the pupil/iris can be detected and how well gaze measurements from several right eye clips coincide.
EstimatedRightGazeHeading	rad	The heading angle of the estimated right gaze direction. (See Section on "Definition of Euler Angles" in the SEP manual).
EstimatedRightGazePitch	rad	The pitch angle of the estimated right gaze direction. (See Section on "Definition of Euler Angles" in the SEP manual).
ASCIIKeyboardState		The ASCII code of the pressed key as an integer.
CalibrationGazeIntersection		Only used to send the selected calibration object to clients. Useful if a calibration point shall be displayed on a screen when performing the gaze calibration.
TaggedGazeIntersection		The position of the current gaze target. It contains the name of the object, the position in world coordinates and the position in object coordinates.
LeftClosestWorldIntersection		The closest intersection of the left eye gaze vector with any of the world objects. The intersection information contains name of object, intersection point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame this integer will be 0, otherwise 1.
LeftAllWorldIntersections		Analogue to LeftClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected. As with LeftClosestWorldIntersection, each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. The difference is that in this case there may be any number of intersections.
RightClosestWorldIntersection		The closest intersection of the right eye gaze vector with any of the world objects. The intersection information contains name of object, intersection point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame this integer will be 0, otherwise 1.

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
RightAllWorldIntersections		Analogue to RightClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected. As with RightClosestWorldIntersection, each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. The difference is that in this case there may be any number of intersections.
FilteredLeftClosestWorldIntersection		The closest intersection of the filtered left gaze vector with any of the world objects. The intersection information contains name of object, intersection point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame this integer will be 0, otherwise 1.
FilteredLeftAllWorldIntersections		Analogue to FilteredLeftClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected.As with ClosestWorldIntersections, each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. The difference is that in this case there may be any number of intersections.
FilteredRightClosestWorldIntersection		The closest intersection of the filtered right gaze vector with any of the world objects. The intersection information contains name of object, intersection point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame this integer will be 0, otherwise 1.
FilteredRightAllWorldIntersections		Analogue to FilteredRightClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected.As with ClosestWorldIntersections, each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. The difference is that in this case there may be any number of intersections.

Table B.1: Descriptions for different Output Data Types

Name	**Unit**	**Description**
EstimatedLeftClosestWorldIntersection		The closest intersection of the estimated left gaze vector with any of the world objects. The intersection information contains name of object, intersection point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame this integer will be 0, otherwise 1.
EstimatedLeftAllWorldIntersections		Analogue to EstimatedLeftClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected. As with ClosestWorldIntersections, each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. The difference is that in this case there may be any number of intersections.
EstimatedRightClosestWorldIntersection		The closest intersection of the estimated right gaze vector with any of the world objects. The intersection information contains name of object, intersection point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame this integer will be 0, otherwise 1.
EstimatedRightAllWorldIntersections		Analogue to EstimatedRightClosestWorldIntersection, but it contains the whole list of intersected world objects. E.g., if the right window of a car is intersected it may be interesting to also find out if the right rear mirror is intersected. As with ClosestWorldIntersections, each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. The difference is that in this case there may be any number of intersections.
LeftBlinkClosingMidTime	100ns	See Section 3.6.4.
LeftBlinkOpeningMidTime	100ns	See Section 3.6.4.
LeftBlinkClosingAmplitude	m	See Section 3.6.4.
LeftBlinkOpeningAmplitude	m	See Section 3.6.4.
LeftBlinkClosingSpeed	m/s	See Section 3.6.4.
LeftBlinkOpeningSpeed	m/s	See Section 3.6.4.
RightBlinkClosingMidTime	100ns	See Section 3.6.4.
RightBlinkOpeningMidTime	100ns	See Section 3.6.4.
RightBlinkClosingAmplitude	m	See Section 3.6.4.

Table B.1: Descriptions for different Output Data Types

Name	Unit	Description
RightBlinkOpeningAmplitude	m	See Section 3.6.4.
RightBlinkClosingSpeed	m/s	See Section 3.6.4.
RightBlinkOpeningSpeed	m/s	See Section 3.6.4.
LeftEyelidState		Detected state of left eyelid. Possible values: "Not Set" (0), "Open" (1), or "Closed" (2).
RightEyelidState		Detected state of right eyelid. Possible values: "Not Set" (0), "Open" (1), or "Closed" (2).
UserMarker		User defined marker, containing some data and the specific time it was received. The marker could, for example, represent a key press on some external experiment equipment. The marker also contains an "Error" field, containing an error code from the Smart Eye error specification. The start of each sub-packet of this type contains a 1 if a marker is received, otherwise 0.
CameraClocks	100ns	Camera hardware provided time stamps of image exposure. Characteristics of these time stamps are camera dependent.

Note

The Rodrigues format is a convenient way to describe rotations: The length of the vector defines the rotation angle about the axis defined by the vector it self. Be careful not to interpret the components of the Rodrigues vector as true rotations about the axes in the corresponding coordinate system. This can only be done if all components are close to zero, otherwise not. This vector can easily be converted to a rotation matrix. Example code to do this is shipped with the system.

Note

There exist two version of these data items, unfiltered and filtered. The filtered data items have the same name but is preceded with Filtered.

Note

There are some regular data items that have estimated values. The estimated data items have the same name but are preceded with Estimated. When using tracking that is not using corneal reflection, the estimated and regular values will display the same value.

Appendix C

Limitations of Smart Eye Pro Editions

C.1 Smart Eye Tracker

C.1.1 Limitation of JSON-RPCs

The following RPC methods are *not* available in SET:

Tracking
startChessboardTracking
stopChessboardTracking

Recording
All methods

Image source
setImageSourceRecording

Playback
All methods

C.1.2 Limitation of Output Data

The following output data ids are *not* available in SET:

```
SEHeadRotationRodrigues
SEHeadNoseDirection
SEHeadUpDirection
SEHeadLeftEarDirection
SEHeadHeading
SEHeadPitch
SEHeadRoll
SEGazeOrigin
SEGazeHeading
SEGazePitch
SELeftEyePosition
SELeftGazeOrigin
SELeftGazeDirection
SELeftGazeHeading
SELeftGazePitch
```

```
SERightEyePosition
SERightGazeOrigin
SERightGazeDirection
SERightGazeHeading
SERightGazePitch
SEFilteredLeftGazeDirection
SEFilteredRightGazeDirection
SEFilteredGazeHeading
SEFilteredGazePitch
SEFilteredLeftGazeHeading
SEFilteredLeftGazePitch
SEFilteredRightGazeHeading
SEFilteredRightGazePitch
SELeftLowerEyelidExtremePoint
SELeftUpperEyelidExtremePoint
SERightLowerEyelidExtremePoint
SERightUpperEyelidExtremePoint
SEEstimatedGazeOrigin
SEEstimatedGazeDirection
SEEstimatedGazeHeading
SEEstimatedGazePitch
SEEstimatedLeftGazeOrigin
SEEstimatedLeftGazeDirection
SEEstimatedLeftGazeHeading
SEEstimatedLeftGazePitch
SEEstimatedRightGazeOrigin
SEEstimatedRightGazeDirection
SEEstimatedRightGazeHeading
SEEstimatedRightGazePitch
SEFilteredEstimatedGazeDirection
SEFilteredEstimatedLeftGazeDirection
SEFilteredEstimatedRightGazeDirection
SEFilteredEstimatedGazeHeading
SEFilteredEstimatedGazePitch
SEFilteredEstimatedLeftGazeHeading
SEFilteredEstimatedLeftGazePitch
SEFilteredEstimatedRightGazeHeading
SEFilteredEstimatedRightGazePitch
```

C.1.3 Limitation of World Model

No limitation.

C.2 Smart Sim

C.2.1 Limitation of JSON-RPCs

No limitation.

C.2.2 Limitation of Output Data

Smart Sim is limited to *only* the following output data ids:

```
SEEyePosition
```

```
SEHeadPositionQ
SEGazeDirection
SEFilteredGazeDirection
SEGazeDirectionQ
SEFilteredGazeDirectionQ
SEClosestWorldIntersection
SEFilteredClosestWorldIntersection
SEFrameNumber
SERealTimeClock
```

C.2.3 Limitation of World Model

World model may only contain objects of type `Screen`.

C.3 Smart Eye XO

C.3.1 Limitation of JSON-RPCs

No limitation.

C.3.2 Limitation of Output Data

No limitation.

C.3.3 Limitation of World Model

No limitation.