Eye Tracking System Instructions

ASL Eye-Trac 6000

Software Development Kit (SDK)

MANUAL VERSION 1.12
9 June 2006

Applied Science Laboratories
An Applied Science Group Company

175 Middlesex Turnpike
Bedford, MA 01730 USA
Tel: (781) 275-4000
Fax: (781) 275-3388
Email: asl@a-s-l.com
Support : techsupport@a-s-l.com
Web site: http://www.a-s-l.com

Table of Contents

1 Introduction to Eye Tracker SDK

ASL Eye Tracker Software Development Kit (SDK) provides access to eye tracker data from third party applications. SDK includes a wide variety of tools and examples showing how to communicate with Eye Tracker using programming languages or stimulus presentation software such as Presentation or E-Prime. Most 3rd party applications can access Eye Tracker using COM servers described in sections 2, 3, 4. The rest of the document describes SDK samples showing how to use Eye Tracker with 3rd party applications such as Matlab, EPrime, Presentation.

1.1 Using COM servers

ASL provides access to Eye Tracker using Microsoft Component (COM) technology. Microsoft Component technology is described in multiple publications, for example in MSDN library at
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/com/htm/comportal_3qn9.asp
or
MSDN Home > MSDN Library > Win32 and COM Development > Component Development > Component Object Model (General)

Product distribution includes 3 Automation servers, each implemented in a separate DLL. The following interfaces are supported:

- **IASLFile (ASLFileLib.DLL)** – read access to eye data files created by Eye Tracker.
- **IASLSerialOutPort2 and IASLSerialOutPort3 (ASLSerialOutLib2.DLL)** – read access to Control Unit Serial Out port
- **IASLControllerPort (ASLControllerLib.DLL)** – read/write access to Control Unit Controller port

ASL product installation includes sample client applications for all supported interfaces. The sample clients are created in Visual Basic and Visual C++ and implement all essential functions of the interfaces. Users should refer to those clients for implementation details.

The sample code below demonstrates how to connect to ASL COM objects using **IASLControllerPort** interface as an example.

C++ Client

```
// Support for CComPtr
#include <atlbase.h>

// Import interface definition
#import  “ASLControllerLib.tlb" no_namespace raw_interfaces_only

// Create COM object
CComPtr<IASLControllerPort> pIASLControllerPort;
HRESULT hr = pIASLControllerPort.CoCreateInstance (__uuidof(ASLControllerPort));

// Use COM object to connect to to Eye Tracker Control Unit via COM1 port
// at baud rate 115200
hr = pIASLControllerPort->Connect(1, 115200);
```

Visual Basic Client

In the Visual Basic menu select Project -> References. In the list of available references find "ASL Controller Library" and select it.

```
' Create COM Object
Dim asl As New ASLControllerPort

' Connect to to Eye Tracker Control Unit via COM1 port at baud rate 115200
asl.Connect 1, 115200
```

Note: Visual Basic client for Serial Port Interface (IASLSerialOutPort2 or IASLSerialOutPort3) can use callbacks from COM server only when it is running from Microsoft Visual Studio environment. Independent executable built from VB code will crash if user tries to read data continuously. This is caused by the fact that VB doesn't support multithreading. The problem is described in Microsoft Knowledge Base article 280512. The workaround provided by Microsoft makes the DLL use approximately 3 times more CPU time, which may create a problem for high-speed systems. If you need a version of the DLL that supports VB clients in continuous reading mode, please contact ASL.

1.2 Sample COM clients supplied by ASL

ASL supplies sample clients for all COM servers described in this manual. The standard ASL software installation installs the source code for the sample clients to
"C:\Program Files\ASL Eye Tracker 6000\SDK\"

The client source code is organized by the software technology and by the interface. For example, the Visual C++ client for ASLSerialOut COM server is located in:
"C:\Program Files\ASL Eye Tracker 6000\SDK\VisualCPP\SerialOutClient\"

The MATLAB clients for both ASLFile and ASLSerialOut interfaces are located in
"C:\Program Files\ASL Eye Tracker 6000\SDK\Matlab\".

The source code is designed to illustrate most of the capabilities of the ASL COM servers. It can be used as a basis for customer's own applications. Please refer to the client code for details.

2 File Interface (IASLFile)

2.1 Data Structures

```
Information from the file header. Returned by the function GetFileInfo() after
opening the file.

struct ASLFile_HeaderData
{
  BSTR eyedat_file_name;
  long version;
  BSTR system_type;
  BSTR monocular_or_binocular;
  BSTR eyd_or_ehd;
  long update_rate;
  BSTR distance_units;
  BSTR angle_units;
  BSTR user_description;
  BSTR creation_time;
  long number_of_segments;
  long number_of_overtimes;
};
```

2.2 Interface Description

```
// Open the file and read header information
// Internal read pointer is positioned to the first record of first segment
// Returns file information and the list of items included in a single record

virtual HRESULT OpenFileForReading(
      BSTR filename/*[in]*/,
      long* segmentCount/*[out]*/,        // Number of data segments in the file
      LPSAFEARRAY* itemNames/*[out]*/,    // names of items included in
                                          // data record (array of BSTR)
      long* itemCount/*[out]*/            // size of itemNames array
      long* overtimeCount/*[out]*/);      // special info for data analysis tools
                                          // (typically not used)

//-----------------------------------------------------------------------

// Close the file after reading is finished
virtual HRESULT CloseFile(void);

//-----------------------------------------------------------------------

// Locate the record with given segment number and video field number
// and position internal read pointer to that record

virtual HRESULT FindRecord (
      long segment_number/*[in]*/,
      long video_field_number/*[in]*/);
```

```
//----------------------------------------------------------------------

// Locate the record with given time relative to segment start
// and position the internal read pointer to that record.
// Function also returns the record video field number (for info only)

virtual HRESULT FindRecordByTime(
      long segment_number/*[in]*/,
      long time_in_msecs/*[in]*/,
      long* video_field_number/*[out,retval]*/);

//----------------------------------------------------------------------

// Read a single record located at the current read pointer,
// and advance the read pointer to the next record.
// Function returns an array of variants ordered the same way as array of names
// returned by OpenFileForReading()

virtual HRESULT ReadDataRecord(
      LPSAFEARRAY* dataItems/*[out]*/,
      long* itemCount/*[out]*/);


//----------------------------------------------------------------------

// Get information about a given segment
// Note: number of segments is returned by OpenFileForReading() as a part of
// ASLFile_HeaderData structure

virtual HRESULT GetSegmentInfo(
      long segment/*[in]*/,          // segment number
      long* numberDataRecords/*[out]*/, // number of records in the segment
      double* startTime/*[out]*/,    // segment start time calculated as number of
                                     // seconds since beginning of the day
      double* stopTime/*[out]*/,     // stop time in the same units as start time
      long* startVideoAnnotationCount/*[out]*/, // initial value of video field #
      long* startVideoAnnotationCountRightEye/*[out]*/); // binocular system only

//Note:
//Segment start and stop time can be translated to hr:min:sec format using
//following calculation:
//    int hours = (int) startTime / 3600;
//    int minutes = (int)(fmod(startTime, 3600.0) / 60.0);
//    double seconds = fmod(startTime, 60.0);


//----------------------------------------------------------------------

// Get information about a given overtime. Overtimes represent missing
// data packets from Control Unit.  Total number of overtimes is returned
// by OpenFileForReading() as a part of ASLFile_HeaderData structure
// Note: typically user doesn’t need this information. Each record contains
// a video field number that already accounts for overtimes

virtual HRESULT GetOvertimeInfo(
      long overtimeNo/*[in]*/,      // overtime number (input)
      long* overtimeCount/*[out]*/, // number of lost data packets
      long* segment/*[out]*/,       // segment number where this overtime occurred
      long* videoFieldCount/*[out]*/, // video field count - see note below
      long* fileFieldCount/*[out]*/); // file field count - see note below
```

Note: Consecutive file field counts (*fileFieldCount* variable) is always incremented by one.
The corresponding video field counts (*videoFieldCount* variable) may be incremented by more than one if any overtimes occurred prior to receiving the latest field (due to any data packet losses from the control unit).
For example:

```
fileFieldCount:   1  2  3  4  5  6  7
videoFieldCount:  1  2  3  6  7  9  10
overtimeCount:    0  0  0  2  0  1  0
```

```
//-----------------------------------------------------------------------

// Get information from the file header (see above description of
// ASLFile_HeaderData structure)

virtual HRESULT GetFileInfo(
      ASLFile_HeaderData* headerData/*out*/);


//-----------------------------------------------------------------------

// The actual update rate of the eye camera can be slightly different from
// the nominal value. For example, 60 Hz pan/tilt camera usually has the
// update rate of 59.94 Hz. This function allows the client to set the
// exact update rate that will be used when calculating records time stamps

virtual HRESULT SetExactUpdateRate(double updateRate/*[in]*/);

//-----------------------------------------------------------------------

// The function GetLastError() returns an error description.
// It should be called after any other interface function returns an error.
// It is especially useful when OpenFileForReading() fails.
// This function clears the error after returning it.

virtual HRESULT GetLastError(BSTR* errorDesc/*[out,retval]*/)
```

2.3 Description of the Data Record

Function *ReadDataRecord(…)* returns an array of VARIANT that represents a single data record from file. The array includes system items (always present) as well as items selected by the user prior to writing to the file. Function *OpenFileForReading(…)* returns an array of names of the items included in the record. A complete list of item names is given below.

Item name	**Description**
System items that are always included	
segment	Segment number
video_field_#	Video field number
time	Record time stamp as string (“hr:min:sec.msec”)

total_secs	Record time stamp as double value: number of seconds since midnight; this value can be translated to hr:min:sec format using following calculation:
	int hours = (int) total_secs / 3600;
	int minutes = (int)(fmod(total_secs, 3600.0) / 60.0);
	double seconds = fmod(total_secs, 60.0);

pupil_recogn	Pupil recognition flag
CR_recogn	Cornea reflection recognition flag
right_pupil_recogn	(Binocular only) right eye pupil recognition flag
right_CR_recogn	(Binocular only) right eye cornea reflection recognition flag
hdtrk_enabled	Head tracker enabled flag
Items selectable by user. Note: in binocular system each item will appear in the selection list twice with prefix left_ or right_ (e.g. left_pupil_diam, right_pupil_diam)	
CU_video_field_#	Video field number from Control Unit; corresponds to the value superimposed on scene image when Scene Video Field Count Annotation is enabled (special hardware required)
XDAT	Value received by Control Unit at parallel data port labeled XDAT
pupil_pos_horz	Horizontal pupil center position (0 at left edge of pupil camera image, 8320 at right)
pupil_pos_vert	Vertical pupil center position (0 at top edge of pupil camera image, 7680 at bottom)
pupil_diam	Pupil diameter measured horizontally (in “pixels”, where width of eye camera field is 260)
pupil_height	Pupil diameter measured vertically (distance between eyelids if they are partially occluding the pupil)
cr_pos_horz	Horizontal cornea reflection position (0 at left edge of pupil camera image, 8320 at right)
cr_pos_vert	Vertical cornea reflection position (0 at top edge of pupil camera

	image, 7680 at bottom)
cr_diam	Cornea reflection diameter
horz_gaze_coord	Horizontal gaze position with respect to the head
vert_gaze_coord	Vertical gaze position with respect to the head
eyeplot_x	(Eyehead Integration only) Point of gaze with respect to the head expressed in Eyehead Scene Plane 0 coordinate system. **Presently disabled**.
eyeplot_y	See explanation for eyeplot_x
Head tracker location (position and orientation of head tracker "sensor" in head tracker coordinate frame)	
hdtrk_X	X coordinate
hdtrk_Y	Y coordinate
hdtrk_Z	Z coordinate
hdtrk_az	Azimuth
hdtrk_el	Elevation
hdtrk_rl	Roll
Eyehead integration data	
EH_scene_number	Scene plane number
EH_horz_gaze_coord	Horizontal point of gaze coordinate in scene plane coordinates (inches or cm from plane origin)
EH_vert_gaze_coord	Vertical point of gaze coordinate
EH_gaze_length	Distance of eye from point of gaze (inches or cm)
EH_eye_location_X	Location of the eye in head tracker coordinate frame
EH_eye_location_Y	
EH_eye_location_Z	
EH_gaze_dir_X	Direction of gaze in head tracker coordinate frame
EH_gaze_dir_Y	
EH_gaze_dir_Z	
Auxiliary sensor location (position and orientation of "auxiliary sensor" in head tracker coordinate frame)	
aux_sensor_X	X coordinate
aux_sensor_Y	Y coordinate
aux_sensor_Z	Z coordinate
aux_sensor_az	Azimuth
aux_sensor_el	Elevation
aux_sensor_rl	Roll
Items available in Binocular system only	
vergence_angle	Angle between left and right eye gaze vectors
verg_gaze_coord_x	Point of gaze location computed using vergence data
verg_gaze_coord_y	

verg_gaze_coord_z_left	
verg_gaze_coord_z_right	

3 Serial Port Interfaces (IASLSerialOutPort2 and IASLSerialOutPort3)

3.1 Introduction

The purpose of these interfaces is to provide convenient read access to the data sent from Eye Tracker Control Unit Serial Out port.

The library ASLSerialOutLib2.DLL supports two interfaces: *IASLSerialOutPort3* and *IASLSerialOutPort2*. The main difference is that newer interface uses E6000.cfg file to obtain information about serial port configuration. The older interface requires the user to enter configuration data. *IASLSerialOutPort3* is the latest interface and is recommended for all new customers. The older interface *IASLSerialOutPort2* is supported for legacy products and also for special needs when greater flexibility is required.

The Eye Tracker Control Unit scales some data items before sending them to the Serial Out port. The goal is to fit them into 16-bit integers while preserving sufficient resolution. Specifically:
point of gaze data (horz_gaze_coord, vert_gaze_coord) is multiplied by 10,
head tracker coordinates are multiplied by 100,
eyehead integration data (point of gaze, eye location) is multiplied by 100,
eyehead integration gaze direction is multiplied by 1000.

The old interface IASLSerialOutPort2::GetDataRecord() returns raw data items as they arrive from Control Unit. The new interface IASLSerialOutPort3::GetScaledData() scales data items and restores their original floating point value. For example, horz_gaze_coord will be divided by 10.

Both *IASLSerialOutPort3* and *IASLSerialOutPort2* use the same event interface. For details please refer to the sample clients included with the SDK.

The SDK distribution includes a number of sample clients written in various programming languages. The source code is provided for all clients. The Visual Basic client (ETSerialPortViewerVB) uses interface IASLSerialOutPort2. The Visual C++ client (ETSerialPortViewer) uses the new interface IASLSerialOutPort3.

The Eye Tracker must be active in order to read from the Serial Out port.

The older interface *IASLSerialOutPort* is no longer supported.

3.2 Enumerations and Data Structures

```
// Enumeration used to describe the type of an item that is returned from the port
typedef enum ASLSerialOutPort_Type
{
  ASL_TYPE_BYTE = 0,
  ASL_TYPE_SHORT = 1,
  ASL_TYPE_UNSIGNED_SHORT = 2,
  ASL_TYPE_LONG = 3,
  ASL_TYPE_UNSIGNED_LONG = 4,
  ASL_TYPE_FLOAT = 5,
  ASL_TYPE_NO_VALUE = 6
} ASLSerialOutPort_Type;
```

3.3 Interface IASLSerialOutPort3 Description

```
interface IASLSerialOutPort3 : IDispatch

// read the configuration file and acquire the port
// Note that "itemNames" array is allocated in the function. Client should release
// the memory by calling SafeArrayDestroy(itemNames).
[id(1), helpstring("method Connect")] HRESULT Connect(
      [in] BSTR cfgFile,      // full path to the .cfg file used by Eye Tracker
      // typically, "C:\Program Files\ASL Eye Tracker 6000\EyeTracking\E6000.cfg"
      [in] long port,         // serial port number
      [in] VARIANT_BOOL eyeheadIntegration,// indicates that Eye Tracker is
                                           // in EyeHead integration mode
      [out] long* baudRate,   // serial port baud rate,
                              // as set by Eye Tracker user interface
      [out] long* updateRate, // eye camera update rate - defines how often
                              // messages are sent in streaming mode
      [out] VARIANT_BOOL* streamingMode,  // streaming or on-demand
      [out] long* itemCount,              // number of data items in the message
      [out] SAFEARRAY(BSTR) * itemNames   // item names (to use as column headers
                                          // when displaying data)
);

//---------------------------------------------------------------------

// Close COM port
[id(2), helpstring("method Disconnect")] HRESULT Disconnect();

//---------------------------------------------------------------------

// Initiate callbacks from COM Server to client - used in streaming mode
[id(3), helpstring("method StartContinuousMode")] HRESULT StartContinuousMode();

//---------------------------------------------------------------------

// Stop callbacks
[id(4), helpstring("method StopContinuousMode")] HRESULT StopContinuousMode();

//---------------------------------------------------------------------
```

```
// Get a single message. Data is returned as an array of Variants.
// This function may be called in both on demand mode and streaming mode
// Control Unit scales some items (e.g. multiplies by 100)
// in order to cast them from 4-byte float to 2-byte integer.
// This function compensates for that and returns original floating point values
// Note that "items" array is allocated in the function. Client should release
// the memory by calling SafeArrayDestroy(items).
[id(5), helpstring("method GetScaledData")] HRESULT GetScaledData(
      [out] SAFEARRAY(VARIANT)* items, // parsed message (items are scaled back
                                       // to their original value)
      [out] long * count,                 // number of data items in the array
      [out] VARIANT_BOOL * available // true indicates that a new record
                                    // has been received,
                                    // false means that no new data has arrived
                                    //since previous function call
);

// Return the description of the last error.
// Should be called after any other function returns an error.
// Note: this function clears the error description
[id(6), helpstring("method GetLastError")] HRESULT GetLastError(
      [out] BSTR* errorDesc         // error description
);
```

3.4 Events Interface

The client must implement this interface. It is used for continuous reading of data in the streaming mode. When a record is received, the COM server will call the function below. Implementation should call GetScaledData(...) to retrieve the message.

Note: The server performs callbacks from a separate thread. It puts some limitations on the C++ client that implements the callbacks. For example, the client code cannot have a pointer to a main client window because it is a part of a separate not window-based thread. Instead client should use a handle to the main window and send it messages. Refer to the sample C++ client (ETSerialPortViewer) for details.

```
dispinterface _IASLSerialOutPort2Events
{
properties:
methods:

// Callback notifies the client that new message has arrived (used in
// streaming mode).
// Client should call GetScaledData(...) to retrieve the message
[id(1), helpstring("method Notify")] HRESULT Notify();
};
```

3.5 Interface IASLSerialOutPort2 Description (for legacy products)

```
interface IASLSerialOutPort2 : IDispatch

// Initialize COM port, define message format and baud rate
[id(1), helpstring("method Connect")] HRESULT Connect(
      [in] long port,         // serial port number
      [in] long baudRate,     // set to 57600, higher rate is not supported
      [in] VARIANT_BOOL streamingMode,
      [in] long updateRate,   // can be 60, 120, 240 or 360
      [in] SAFEARRAY(long) *  itemTypes // type of each item in the message
);

//-----------------------------------------------------------------------

// Close COM port
[id(2), helpstring("method Disconnect")] HRESULT Disconnect();

//-----------------------------------------------------------------------

// Initiate callbacks from COM Server to client - used in streaming mode
[id(3), helpstring("method StartContinuousMode")] HRESULT StartContinuousMode();

//-----------------------------------------------------------------------

// Stop callbacks
[id(4), helpstring("method StopContinuousMode")] HRESULT StopContinuousMode();

//-----------------------------------------------------------------------

// Get a single message. Data is returned as an array of Variants.
// Note that array is allocated in the function and client should release
// the memory by calling SafeArrayDestroy(items).
// Note: this function may be called in both on demand mode and streaming mode
[id(5), helpstring("method GetDataRecord")] HRESULT GetDataRecord(
      [out] SAFEARRAY(VARIANT)* items, // parsed message
      [out] long * count,              // number of data items in the array
      [out] VARIANT_BOOL * available   // true indicates that a new record
                                       // has been recieved,
                                       // false means that no new data has
                                       // arrived since previous function call
);


// Return the description of the last error.
// Should be called after any other function returns an error.
// Note: this function clears the error description
[id(6), helpstring("method GetLastError")] HRESULT GetLastError(
      [out] BSTR* errorDesc         // error description
);
```

4 Controller Port Interface (IControllerPort)

4.1 Changes in Version 2.0

Version 2.0 was brought closer to real Eye Tracker operation. The obsolete functions that are not used were deleted from the interface. Also new functions were included reflecting the changes made to the Eye Tracker since the release of this interface.

Deleted Functions:

get_statistics(ASLController_Statistics* stats)
Also deleted: struct ASLController_Statistics
set_video_input_source(long value)
get_video_input_source(long* value)
set_eye_video_source(long value)
get_eye_video_source(long* value)
reset_system()
set_camera_power_mode(long value)
get_camera_power_mode(long* value)
set_ptc_relative_position(long pan, long tilt)
set_pupil_cursor_intensity(long value)
get_pupil_cursor_intensity(long* value)
set_cr_cursor_intensity(long value)
get_cr_cursor_intensity(long* value)
set_horizontal_breaks_mode(long value)
get_horizontal_breaks_mode(long* value)
get_pattern_recognition_results(…)
sample_mirror_calibration_data(…)
set_mirror_calibration_point_data(…)
get_mirror_calibration_point_data(…)
compute_mirror_transform_coefficients()
set_eye_monitor_white_level(long value)
get_eye_monitor_white_level(long* value)
set_scene_monitor_white_level(long value)
get_scene_monitor_white_level(long* value)
set_vitc_level(long value)
get_vitc_level(long* value)
set_pupil_diameter_offset(long value)
get_pupil_diameter_offset(long* value)
set_pupil_diameter_gain(long value)
get_pupil_diameter_gain(long* value)
set_pattern_recognition_mode(long value)
set_transform_mode(long value)
start_edge_data_collection()
stop_edge_data_collection()
set_mht_port(…)

start_mht(…) // *Note: use reset_mht() to start the head tracker*
boresight_mht(long* completion_code)
get_edge_data(long* p, long* c)
get_edge_data_buffer(long segment, unsigned_char* outbuffer)
get_ptc_calibration_mode(long* value)

New Functions:

set_ptc_auto_focus_setting(VARIANT_BOOL value)
set_ptc_focus_setting(long value)
get_ptc_focus_value(long* value)
get_ptc_focus_range(long* minValue, long* maxValue)
set_ptc_reset()
get_ptc_parameter(long id, long* value)

4.2 Data Structures

```
struct ASLController_EyeheadGeneralParameters
{
  float sensor_to_eye_vector__head[3];
  float horz_final_unit_offset;
  float vert_final_unit_offset;
  float horz_final_unit_gain;
  float vert_final_unit_gain;
  long number_of_defined_scenes;
};

struct ASLController_EyeheadSceneParameters
{
  float point_a_coordinate__scn[2];
  float point_b_coordinate__scn[2];
  float point_c_coordinate__scn[2];
  float gimbal_pointer_magnitude;
  float source_to_point_a_magnitude;
  float source_to_point_b_magnitude;
  float source_to_point_c_magnitude;
  float point_a_direction[3];
  float point_b_direction[3];
  float point_c_direction[3];
  float point_a_angles[3];
  float point_b_angles[3];
  float point_c_angles[3];
  float sensor_wand_tip__sen[3];
  float source_to_point_a__trn[3];
  float source_to_point_b__trn[3];
  float source_to_point_c__trn[3];
  float source_to_scene_normal_magntude;
  float source_to_scene_normal__trn[3];
  float source_to_scene_normal_angles[3];
  float scene_normal_offset__scn[2];
  float plane_equation_coefficient__trn[3];
  float plane_equation_constant__trn;
  float source_to_plane_origin__trn[3];
  float top_scene_boundary_limit__scn;
  float bottom_scene_boundry_limit__scn;
  float left_scene_boundary_limit__scn;
  float right_scene_boundary_limit__scn;
  float manual_scene_plane_offset__scn[2];
  byte abc_method;
};

struct ASLController_EyeheadSceneEquationValues
{
  float source_to_point_a__trn[3];
  float source_to_point_b__trn[3];
  float source_to_point_c__trn[3];
  float source_to_scene_normal_magntude;
  float source_to_scene_normal__trn[3];
  float source_to_scene_normal_angles[3];
  float scene_normal_offset__scn[2];
  float plane_equation_coefficient__trn[3];
  float plane_equation_constant__trn;
  float source_to_plane_origin__trn[3];
};

struct ASLController_Results
{
  long frame_number;
  long xdat_value;
  VARIANT_BOOL pupil_center_found;
  long pupil_center_horz;
  long pupil_center_vert;
  long pupil_diameter;
  long pupil_height;
  VARIANT_BOOL cr_center_found;
  long cr_center_horz;
  long cr_center_vert;
  long cr_diameter;
  long scene_x;
  long scene_y;
  float eyeplot_x;
  float eyeplot_y;
  long mht_x_position;
  long mht_y_position;
  long mht_z_position;
  long mht_azimuth_angle;
  long mht_elevation_angle;
  long mht_roll_angle;
  long final_intersecting_scene_plane;
  float actual_point_of_gaze_x_cp;
  float actual_point_of_gaze_y_cp;
  float actual_point_of_gaze_x_ip;
  float actual_point_of_gaze_y_ip;
  float actual_point_of_gaze_magnitude;
  long auto_tracking_mode;
  long source_to_eye__trn[3];
  long eye_to_pog__trn[3];
  long gaze_res_factor;
  long content_flags;
  long auxs_x_position;
  long auxs_y_position;
  long auxs_z_position;
  long auxs_azimuth_angle;
  long auxs_elevation_angle;
  long auxs_roll_angle;
};

struct ASLEyeheadSubjectInfo
{
  float cal_mht_head_position__trn[3];
  float bs_mht_head_position__trn[3];
  float cal_angles[3];
  float cal_bs_angles[3];
  float current_bs_angles[3];
  float current_sensor_to_eye__head[3];
  float cal_sensor_to_eye__head[3];
};
```

4.3 Interface Description

```
interface IASLControllerPort: IDispatch

HRESULT Connect([in] long comPort, [in] long baudRate );

HRESULT Disconnect( void );

HRESULT set_scene_video_source([in] long value );
// input values
// 0 - AUTO
// 1 - REMOTE_SCENE_CAMERA
// 2 - HELMET_SCENE_CAMERA
// 3 – BOTH

HRESULT get_scene_video_source([out, retval] long * value );
//Note: see set_scene_video_source for the list of possible values

HRESULT set_helmet_illuminator_level([in] long value );
//input value must be between 1 and 15

HRESULT get_helmet_illuminator_level([out] long * value );
//Note: see set_helmet_illuminator_level for the output value range

HRESULT set_pupil_threshold([in] long value );
//input value must be between 0 and 255

HRESULT get_pupil_threshold([out, retval] long * value );
//Note: see set_pupil_threshold for the output value range

HRESULT set_discrimination_mode([in] long value );
// input values
// 0 - MANUAL
// 1 – AUTO

HRESULT set_corneal_reflection_threshold([in] long value );
//input value must be between 0 and 255

HRESULT get_corneal_reflection_threshold([out, retval] long * value );
//Note: see set_corneal_reflection_threshold for the output value range

HRESULT set_pupil_mode([in] long value );
// input values
// 0 - BRIGHT
// 1 - DARK

HRESULT get_pupil_mode([out, retval] long * value );
//Note: see set_pupil_mode for the list of possible values

HRESULT get_operational_status([out, retval] unsigned short * status );
//output values
//0 – OFFLINE
//1 – ONLINE

HRESULT get_system_info([out] BSTR * version,
      [out] long * pattern_recognition_algorithm );

HRESULT set_system_type([in] long value );
// input values
// 0 - HELMET_SYSTEM,
// 1 - REMOTE_W_PAN_TILT_SYSTEM,
// 2 - REMOTE_W_PAN_TILT_MHT_SYSTEM,
// 3 - REMOTE_NO_MIRROR_SYSTEM,
// 4 - REMOTE_W_MIRROR_SYSTEM,
// 5 - REMOTE_W_MIRROR_EHT_SYSTEM,
// 6 - REMOTE_W_MIRROR_MHT_SYSTEM,
// 7 - REMOTE_W_MIRROR_EHT_MHT_SYSTEM,
// 8 - EYE_ONLY_SYSTEM

HRESULT get_system_type([out, retval] long * value );
//Note: see set_system_type for the list of possible values


HRESULT set_illuminator_power_mode([in] long value );
// input values
// 0 - OFF,
// 1 - ON

HRESULT get_illuminator_power_mode([out, retval] long * value );
//Note: see set_illuminator_power_mode for the list of possible values

HRESULT set_eye_camera_update_rate([in] long value );
// input values
// 50, 60, 120, 240, 360

HRESULT get_eye_camera_update_rate([out, retval] long * value );
//Note: see set_eye_camera_update_rate for the list of possible values

HRESULT set_scene_camera_update_rate([in] long value );
// input values
// 50, 60

HRESULT get_scene_camera_update_rate([out, retval] long * value );
//Note: see set_scene_camera_update_rate for the list of possible values

HRESULT set_tracking_mode([in] long value );
// input values
// 0 - MANUAL,
// 1 - AUTO

HRESULT set_ptc_exposure_mode([in] long value );
// input values

// input values
// Sony Pan/Tilt camera (Eye Tracker 5000)
//    AUTO = 0x00,
//    MANUAL = 0x03, - recommended setting
//
// Canon Pan/Tilt camera (Eye Tracker 6000)
//    AUTO = 0x32,
//    MANUAL = 0x33,
// Canon Pan/Tilt camera (Eye Tracker 6000)
VCC4_EM_AUTO = 0x32,
VCC4_EM_MANUAL = 0x33,

HRESULT set_ptc_shutter_setting([in] long value );

HRESULT set_ptc_iris_setting([in] long value );

HRESULT set_ptc_gain_setting([in] long value );

HRESULT set_ptc_zoom_setting([in] long value );

HRESULT set_ptc_absolute_position([in] long pan, [in] long tilt );

HRESULT set_distance_from_ptc_to_mht_transmitter([in] float distance );

HRESULT compute_ptc_to_mht_transmitter_vector( void );

HRESULT get_ptc_to_mht_transmitter_vector(
      [out] float * x, [out] float * y, [out] float * z );

HRESULT set_ptc_to_mht_transmitter_vector(
      [in] float x, [in] float y, [in] float z );

HRESULT set_ptc_mht_calibration_data(
      [in] long pnum, [in] float x, [in] float y, [in] float z,
      [in] long pan, [in] long tilt, [in] long focus );

HRESULT compute_ptc_mht_transform_coefficients(
      [out, retval] long * completionCode );

HRESULT get_ptc_mht_transform_coefficients(
      [out] float * azimuth1, [out] float * azimuth2,
      [out] float * elevation1, [out] float * elevation2,
      [out] float * focus1, [out] float * focus2 );
```

```
HRESULT compute_ptc_mht_sensor_to_eye_vector(
      [out, retval] long * completionCode );

HRESULT get_ptc_mht_sensor_to_eye_vector(
      [out] float * x, [in] float * y, [in] float * z );

HRESULT set_ptc_mht_sensor_to_eye_vector(
      [in] float x, [in] float y, [in] float z );

HRESULT set_mht_sensor_to_eye_offset([in] float offset );

HRESULT set_scene_cursor_mode([in] long value );
// input values
// 0 - TARGET_CONTROLLED,
// 1 - HOST_CONTROLLED,

HRESULT set_pog_indicator_type([in] long value );
// input values
// 0 - CROSSHAIR,
// 1 - CURSOR

HRESULT get_pog_indicator_type([out, retval] long * value );
//Note: see set_pog_indicator_type for the list of possible values

HRESULT set_pog_indicator_intensity([in] long value );

HRESULT get_pog_indicator_intensity([out, retval] long * value );

HRESULT set_scene_cursor_position([in] long x, [in] long y );

HRESULT get_scene_cursor_position([out] long * x, [out] long * y );

HRESULT set_scene_camera_mode([in] long value );
// input values
// 0 - REFLECTED,
// 1 - DIRECT

HRESULT get_scene_camera_mode([out, retval] long * value );
//Note: see set_scene_camera_mode for the list of possible values

HRESULT set_scene_target_point_position(
      [in] long pnum, [in] long x, [in] long y );

HRESULT get_scene_target_point_position(
      [in] long pnum, [out] long * x, [out] long * y );

HRESULT sample_eye_calibration_data(
      [out] float * h1, [out] float * v1, [out] float * h2, [out] float * v2 );

HRESULT sample_eye_calibration_data_ex(
      [in] long trimPercent,
      [out] float * h1,
      [out] float * v1,
      [out] float * h2,
      [out] float * v2,
      [out] float * h1q,
      [out] float * v1q );
```

```
HRESULT set_eye_calibration_point_data(
      [in] long pnum,
      [in] float h1, [in] float v1,
      [in] float h2, [in] float v2 );

HRESULT get_eye_calibration_point_data([
      in] long pnum,
      [out] float * h1, [out] float * v1,
      [out] float * h2, [out] float * v2 );

HRESULT compute_scene_transform_coefficients( void );

HRESULT get_eyehead_subject_info([out] ASLEyeheadSubjectInfo * info );

HRESULT set_data_acquisition_mode([in] long value );
//input values
// 0 - REQUEST
// 1 - CONTINUOUS
// 2 - NONE

HRESULT get_data_acquisition_mode([out, retval] long * value );
//Note: see set_data_acquisition_mode for the list of possible values

HRESULT set_eye_position_average_size([in] long size );

HRESULT set_eye_position_offset([in] long x, [in] long y );

HRESULT set_eye_calibration_type([in] long value );
//input values
// 0 - NINE POINTS
// 1 - SEVENTEEN POINTS

HRESULT set_eyehead_units([in] long value );
//input values
// 0 - ENGLISH
// 1 - METRIC

HRESULT set_eyehead_transform_mode([in] long value );
//input values
// 0 - DISABLED
// 1 - ENABLED

HRESULT set_eyehead_test_mode([in] long value );
//input values
// 0 - DISABLED
// 1 - ENABLED

HRESULT set_eyehead_general_parameters(
      [in] ASLController_EyeheadGeneralParameters value);

HRESULT set_eyehead_target_point_position(
      [in] long pnum, [in] float h, [in] float v );

HRESULT get_eyehead_target_point_position(
      [in] long pnum, [out] float * h, [out] float * v );
HRESULT set_eyehead_scene_parameters(
      [in] long snum, [in] ASLController_EyeheadSceneParameters value);

HRESULT calculate_eyehead_scene_equation_values(
      [in] long snum, [out] long * completion_code );

HRESULT get_eyehead_scene_equation_values(
      [in] long snum, [out] ASLController_EyeheadSceneEquationValues * value);

HRESULT perform_eyehead_boresight(
      [out] float * position_x,
      [out] float * position_y,
      [out] float * position_z,
      [out] float * angle_az,
      [out] float * angle_el,
      [out] float * angle_rl,
      [out, retval] long * completion_code );

HRESULT calculate_eyehead_subject_dependant_values( void );


HRESULT set_ssc_transform_mode([in] long value );
//input values
// 0 - DISABLED
// 1 – ENABLED

HRESULT set_ssc_scene_cursor_mode([in] long snum, [in] long value );
//input values
// 0 - DISABLED
// 1 - ENABLED

HRESULT set_ssc_scene_monitor_point(
      [in] long snum, [in] long pnum, [in] long x, [in] long y );
HRESULT get_ssc_scene_monitor_point(
      [in] long snum, [in] long pnum, [out] long * x, [out] long * y );
HRESULT set_ssc_scene_plane_point(
      [in] long snum, [in] long pnum, float h, float v );

HRESULT set_ssc_coefficients([in] long snum, [in] float * h, [in] float * v );

HRESULT get_ssc_coefficients([in] long snum, [out] float * h, [out] float * v );

HRESULT compute_ssc_coefficients(
      [in] long snum, [out, retval] long * completion_code );

HRESULT set_mht_system_type([in] long type,
      [in] float offset_x, [in] float offset_y, float [in] offset_z);
//input values
//0 - POLHEMUS_3_SPACE,
//1 - POLHEMUS_FASTRAK,
//2 - ASCENSION_FLOCK,
//3 - ASCENSION_AHPOMS,
//4 - ASCENSION_XRNG,
//5 - ASCENSION_FLOCK_MASTER,
//6 - ASCENSION_FLOCK_SYNC,
//7 - ASCENSION_FLOCK_MASTER_SYNC,
//8 - ASCENSION_DUAL_SENSOR
//9 - ASCENSION_FLOCK_MONITOR_ONLY
```

```
//10- ASCENSION_LASERBIRD
//11- VIDEO_HEAD_TRACKER_VHT_1
//12- NDI_OPTOTRACK
//13- VIDEO_HEAD_TRACKER_VHT_2

// offsets are typically set to 0

HRESULT stop_mht([out, retval] long * completion_code );

HRESULT reset_mht([out, retval] long * completion_code );
// Note: use reset_mht command to start the head tracker

HRESULT sample_mht_values(
      [out] float * position_x,
      [out] float * position_y,
      [out] float * position_z,
      [out] float * angle_az,
      [out] float * angle_el,
      [out] float * angle_rl,
      [out, retval] long * completion_code );

HRESULT set_analog_output_mode([in] long value );
//input values
// 0 - TARGET CONTROLLED
// 1 - HOST CONTROLLED

HRESULT set_serial_data_out_format_type([in] long value );
//input values
//0 -   NORMAL_ON_DEMAND
//1 -   NORMAL_EYEHEAD_ON_DEMAND
//2 -   TARGET_POINT_ON_DEMAND
//3 -   APPEND_RAW_DATA_ON_DEMAND
//128 - NORMAL_STREAM
//129 - NORMAL_EYEHEAD_STREAM
//130 - TARGET_POINT_STREAM
//131 - APPEND_RAW_DATA_STREAM

HRESULT set_eye_position_blink_filter_value([in] long value );

HRESULT get_results(
      [out] ASLController_Results * results, [out, retval] VARIANT_BOOL * valid );

HRESULT get_interface_status([out, retval] long * value );
//output values
//0 - OFFLINE
//1 - ONLINE

HRESULT updateComm( void ); // Read current results from Control Unit

HRESULT output_target_point([in] long point_number );

HRESULT set_video_annotation_mode([in] long value );
//input values
// 0 - ENABLE
// 1 - DISABLE

HRESULT set_pt_mht_rotation_limit_mode([in] VARIANT_BOOL value );
HRESULT set_pupil_only_mode([in] VARIANT_BOOL value );

HRESULT set_ptc_calibration_mode([in] VARIANT_BOOL value );

HRESULT set_ptc_model([in] long value );
//input values
// 0 - SONY_EVI
// 1 - CANON_VCC4

HRESULT set_ptc_constants([in] unsigned char * values );

HRESULT set_camera_communication_mode ([in] long value );
//input values
// 0 – REMOTE_CONTROL,
// 1 – SERIAL (COMPUTER)

//////////////////////////////////////////////////
// Added in Version 2.0

HRESULT set_ptc_auto_focus_setting([in] VARIANT_BOOL value );

HRESULT set_ptc_focus_setting([in] long value );

HRESULT get_ptc_focus_value([out] long * value );

HRESULT get_ptc_focus_range([out] long * minValue, [out] long * maxValue );

HRESULT set_ptc_reset( void );

HRESULT get_ptc_parameter([in] long id, [out, retval] long * value );
//input values (id)
// 0 - PTC_EXPOSURE_MODE,
// 1 - PTC_SHUTTER_SETTING,
// 2 - PTC_IRIS_SETTING,
// 3 - PTC_GAIN_SETTING,
// 4 - PTC_ZOOM_SETTING,
// 5 - PTC_AUTO_FOCUS_SETTING,
// 6 - PTC_FOCUS_SETTING,
```

5 Using Eye Tracker with Matlab

MATLAB is a high-level language and interactive environment that enables
you to perform computationally intensive tasks faster than with traditional programming languages. It is developed by MathWorks (http://www.mathworks.com/products/matlab).

MATLAB has interfaces that enable it to interact with Microsoft COM objects. Therefore it can use pretty much all functions and interfaces described in the previous sections.

ASL MATLAB SDK includes sample scripts that are installed to C:\Program Files\ASL Eye Tracker 6000\SDK\Matlab. They show how to access Eye Tracker files and Serial Out data from MATLAB. *You will need MATLAB version 7.00 (R14) or higher* in order to execute those samples.

File **ASLFileSample.m** demonstrates how to read eye data files created by ASL Eye Tracker. The sample eye data file is also provided. The script opens the file and reads one data record.

File **ASLSerialOutSample2.m** demonstrates how to read real time data from ASL Eye Tracker Serial Out port. It connects to the serial port using IASLSerialOutPort3 interface and reads two messages.

In order to run **ASLSerialOutSample2.m,** first connect Matlab computer with Eye Tracker control unit as described in Appendix: Testing Serial Out Connection. From the Eye Tracker menu select Configure-> Configure -> Serial Out Port Settings and **uncheck** “Sreaming Mode”. Then start the script and be sure to change Matlab current directory to the location of the script (Matlab will prompt you). If you encounter an error during script execution, you can get more information by typing in the Matlab command window **s.GetLastError**.

If you want to send different information from the eye tracker, first configure serial port using eye tracker menu (Configure -> Serial Out Port Settings), then copy eye tracker configuration file E6000.cfg (or eyetrac6000net.xml if you are using .Net user interface) to your Matlab computer. Eye tracker configuration file is located in C:\Program Files\ASL Eye Tracker 6000\EyeTracking. Edit Matlab script to refer to the correct file. You should include the whole path to the file.

SDK also inludes .log files showing results of the execution of each samle script.

6 Using Eye Tracker with Presentation

ASL Eye Tracker SDK includes special software that allows for entering real time eye position data in into Presentation program. **Presentation** is a high-precision program for stimulus delivery and experimental control for behavioral and physiological experiments (http://nbs.neuro-bs.com/presentation).

6.1 Sending data from Presentation to Eye Tracker (XDAT).

If you need to synchronize eye data recorded by the Eye Tracker with your Presentation experiment, you can send an integer value from Presentation to Eye Tracker. This value will be recorded and can be later used in the analysis of collected data. In order to do that:

1. Find parallel cable supplied by ASL that is labeled **XDAT**. With this cable connect Printer port (or any other parallel port) of you Presentation computer with XDAT port on the back of Eye Tracker Control Unit.
2. Start the Eye Tracker.
3. From your Presentation script right an integer value to the Printer port. Eye Tracker user interface should display it as “XDAT value”. This value will be automatically recorded to eye data file. ASL analysis tool *Eyenal* can use XDAT value when analyzing data (refer to Eyenal manual for details).

Note that no special software is required for reading and recording XDAT value in the Eye Tracker. You will however need to write Presentation script that writes a value.

Refer to Presentation documentation for description how to write a value to the Printer/Parallel port (*Presentation Help: Help Guide: Hardware Interfacing: Port Output*).

6.2 Sending data from Eye Tracker to Presentation.

Using ASL sample experiment

1. **Set up and test serial connection** between Eye Tracker Serial Out port and Presentation computer as described in Appendix: Testing Serial Out Connection.

2. **Register Eye Tracker extension**. Start Presentation and from the menu select *Tools -> Extension Manager*. Click "*Select Extension File*" and navigate to **PresentationLib.dll** (typically in "C:\Program Files\ASL Eye Tracker 6000\DLL\"). In the "Resister As" field enter "**ASLEyeTracker**" (your script will use this name when it creates the eye tracker object). Click "*Register Extension*".

3. **Open the sample experiment.** Eye Tracker installation puts the sample to C:\Program Files\ASL Eye Tracker 6000\SDK\Presentation\.
   Open the script (eye_tracker_test.pcl) in the Presentation Editor and edit it to set the correct port number. In order to do it, locate the line

`tracker.send_string( "port=1" );`
(probably, line 22) and enter the number of the COM port to which you connected the serial cable in Step 1. Note that COM1 corresponds to `"port=1"`.

4. **Run the experiment**. Start Eye Tracker by pressing "T", "D", "U". You should see the eye data. Press "Q" to exit.

You can copy portions of the script that enables Eye Tracker to you own experiment.

Notes on writing PCL script

The Eye-Tracker commands available to Presentation script are described in the Presentation documentation (References -> PCL Reference -> Eye Tracking Types).

ASL Eye Tracker supports only eye_position_data *and* pupil_data.

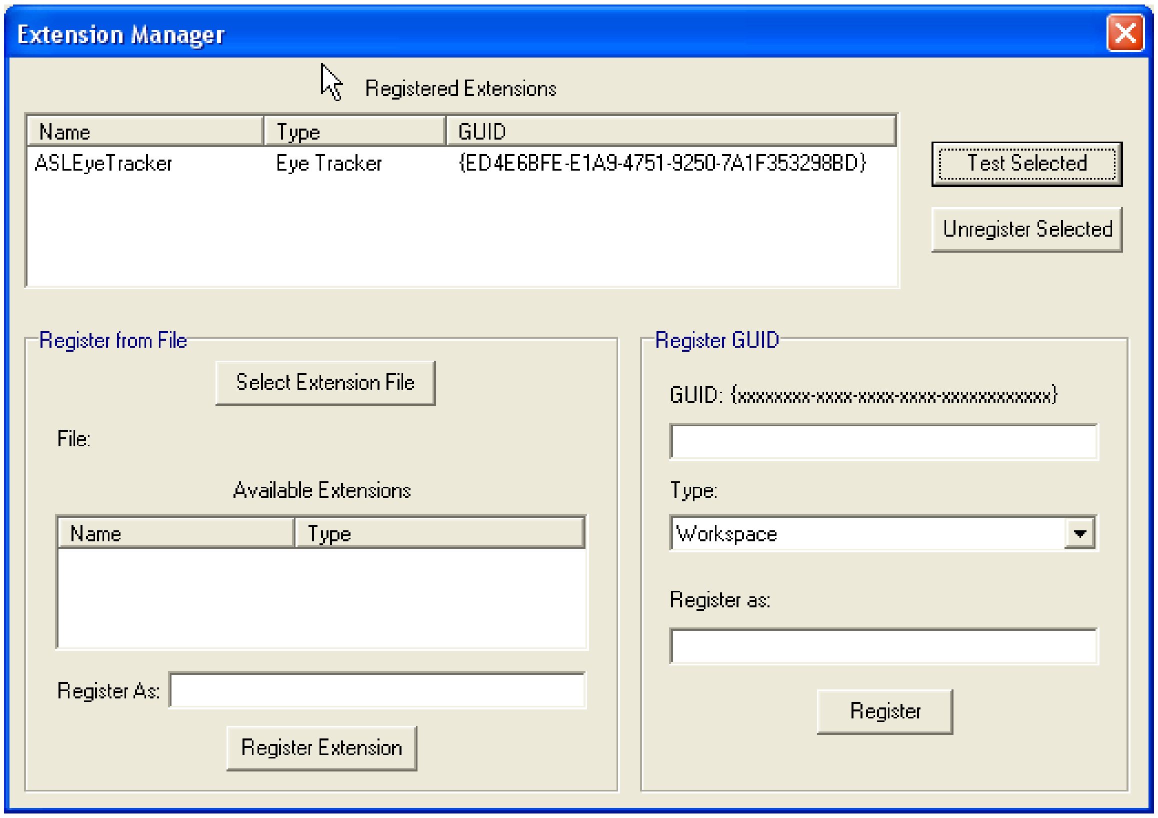
The timestamps are assigned to data in the PC when the Presentation receives the message from the Eye Tracker. Eye Tracker user interface assigns the time stamps independently. Besides, Presentation and Eye Tracker use different zero points for time stamps. It is recommended to use XDAT signal as described in section 6.1 to synchronize your presentation records with Eye Tracker records.

Before PCL script issues start_tracking(), it must tell the Eye Tracker which serial port to read. It is accomplished by issuing the command send_string("port=N") where N is a COM port number, i.e.: tracker.send_string("port=1").

Testing with Presentation Extension Manager

You can use Extension Manager to test Eye Tracker – Presentation link. In order to do that:

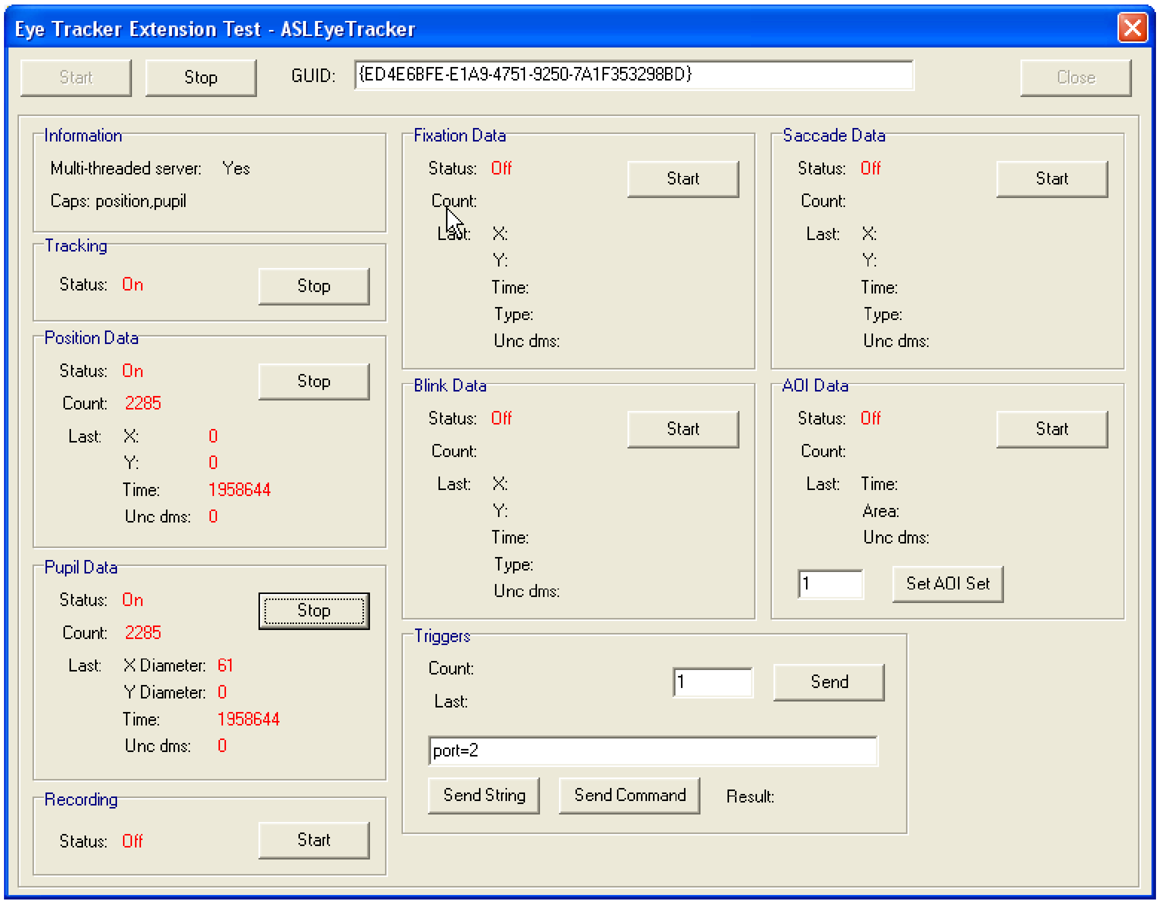
1. Execute steps 1-5 as described in the section **Error! Reference source not found.**.
2. From Presentation menu select Tools->Extension Manager
3. From the list of Registered Extensions select ASLEyeTracker and click "Test Selected"



Name	Type	GUID
ASLEyeTracker	Eye Tracker	{ED4E6BFE-E1A9-4751-9250-7A1F353298BD}

Type: Workspace

4. On the *Eye Tracker Extension Test* dialog press *Start*.
5. On the bottom left corner of the dialog enter "port=.." and press *Send String* (see screen capture below)
6. Start *Tracking*, *Position Data* and *Pupil Data*. You should see the same values as displayed on the Eye Tracker user interface


Eye Tracker Extension Test - ASLEyeTracker

GUID: {ED4E6BFE-E1A9-4751-9250-7A1F353298BD}

Information
Multi-threaded server: Yes
Caps: position,pupil

Tracking
Status: On

Position Data
Status: On
Count: 2285
Last: X: 0
Y: 0
Time: 1958644
Unc dms: 0

Pupil Data
Status: On
Count: 2285
Last: X Diameter: 61
Y Diameter: 0
Time: 1958644
Unc dms: 0

Recording
Status: Off

Fixation Data
Status: Off
Count:
Last: X:
Y:
Time:
Type:
Unc dms:

Blink Data
Status: Off
Count:
Last: X:
Y:
Time:
Type:
Unc dms:

Saccade Data
Status: Off
Count:
Last: X:
Y:
Time:
Type:
Unc dms:

AOI Data
Status: Off
Count:
Last: Time:
Area:
Unc dms:
1

Triggers
Count:
Last:
1
port=2
Result:

7 Using Eye Tracker with EPrime

EPrime is a graphical experiment generator for Windows. It is developed jointly by PsyScope and Psychology Software Tools (http://www.pstnet.com/products/e-prime/).

ASL EPrime SDK installs to "C:\Program Files\ASL Eye Tracker 6000\SDK\EPrime\ ".

File **ASL_Utilities.txt** contains ASL EPrime library. The library allows an EPrime experiment to easily interact with an ASL eye tracker. Refer to this file for detailed description of each function.

The SDK also includes sample EPrime experiments that demonstrate how to use library functions.

File **xdat_test.es** demonstrates sending XDAT signal to Eye Tracker using XDAT cable connecting Printer port on EPrime computer with XDAT port on the Eye Tracker control unit (cable is supplied by ASL). This signal can be used to start and stop data recording and to synchronize Eye Tracker data file with EPrime records. Refer to Eye Tracker manual on the details of using XDAT signal.

File **ASL_test.es** demonstrates the functions of the ASL E-Prime library that scale eye tracker data to the VGA coordinate space and read real time eye tracker data during an EPrime experiment.

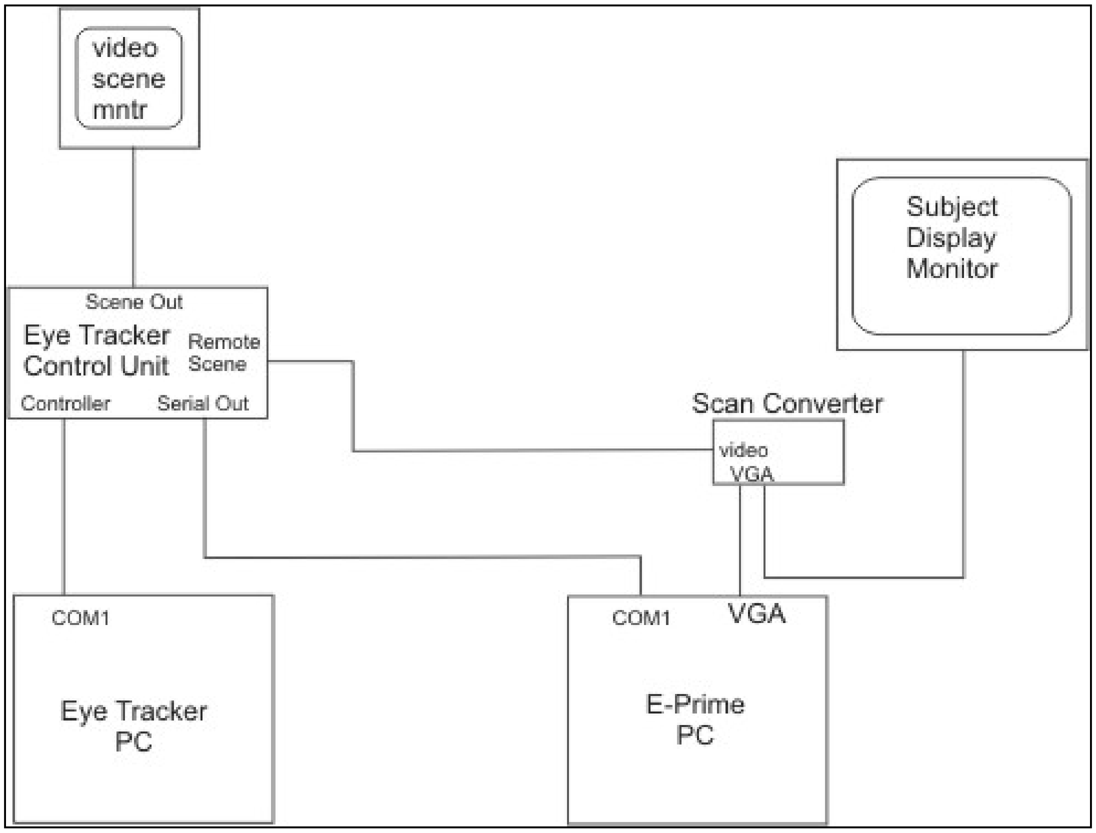
In order to run **ASL_test.es,** frst connect EPrime computer with Eye Tracker control unit as described in Appendix: Testing Serial Out Connection.

It is also assumed that a scan converter is connected to the E-Prime computer, providing a video signal to "Remote Scene" on the ASL control unit, and that "Scene Out" on the control unit is connected to a video monitor (or frame grabber on the Eye tracker PC) so that there is a scene monitor image showing the E-Prime computer screen. See the diagram on the next page.

Put the eye tracker in Set Target mode so that you can control the Point of Gaze (POG) cursor with the mouse. Load the ASL_test.es file in E-Studio (File>Open). Be sure Serial data is enabled on the E-Prime experiment device tab, and that it is set to the proper COM port, and to 57600 baud. On the Eye Tracker user interface, open the Serial Out Port Settings dialog (on the Configuration menu), click "Restore Defaults" and put a check in the box labeled "Streaming Mode". Click OK.

Click the "Run" button on E-Studio. E-Prime will ask for a subj. no. Enter any integer. E-Prime will then ask for a session no. Again, enter any integer. Assuming you are running the test for the first time, E-Prime will then note that "VGA configuration file is not detected". Press the "Recalculate" button. E-Prime will display a target point near the upper left part of the screen. Look at the scene monitor. Use the mouse on the Eye Tracker PC to move the eye tracker POG cursor over the target that E-Prime has drawn, and press <Space Bar> on the EPrime computer keyboard. E-Prime will now draw a target in the lower right. Again use the Eye Tracker PC mouse to put the POG cursor over the E-Prime e target and press <Space Bar> on the E-Prime computer. This procedure has enabled EPrime to compute a transform between Eye Tracker and VGA coordinates (saved in the "VGA configuration file"). EPrime should now start to draw a cursor that moves with the Eye Tracker cursor.

The experiment will exit automatically if coordinate values received from the Eye tracker are zero. To end the E-Prime test, either exit from the “Set Target” mode on the Eye Tracker PC (assuming a pupil is not being recognized, eye tracker point of gaze coordinates will become zero), or type <Cntrl><Alt><Shift> on the E-Prime PC keyboard.



Important connections for running E-Prime ASL interface test (ASL_test.es)

8 Appendix: Testing Serial Out Connection

Third party applications can read eye tracking data in real time using Eye Tracker Serial Out port. This section explains how to set up and test this connection. The procedure described below should be executed before running any third party application, such as the ones described in this manual (Matlab, EPrime etc.).

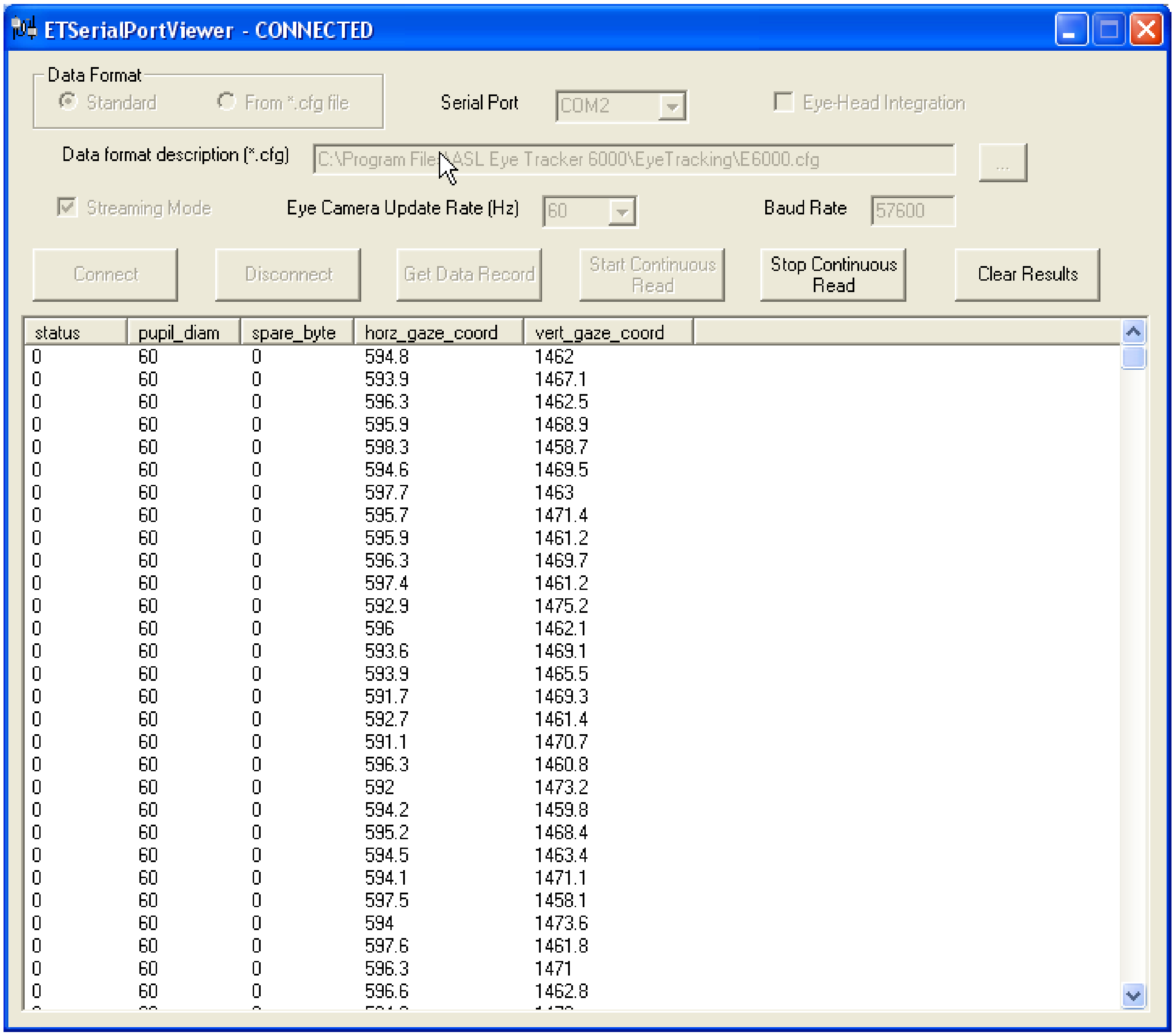
1. **Run a cable** between Serial Out port (located on the back of the Eye Tracker Control Unit) and Serial Port on the PC that runs 3rd party application.
   Note: Beware that serial cables come in two types: “Modem” and “Null Modem”. The appropriate “Serial Out” cable should be “Modem”. It is identical to the cable that connects Eye Tracker “Controller” port to the Eye Tracker PC. To verify that, switch the cables and verify that eye tracker application on PC is on-line (you may have to re-start the eye tracker program on PC).

2. **Install Eye Tracker SDK** on the PC that runs 3rd party application. You can achieve it by simply installing the Eye Tracker from CD. If you don't want to install unnecessary components select Custom Installation and unselect all components except SDK.

3. **Start Eye Tracker** (on the Eye Tracker PC) and make sure that it is using **standard streaming serial out format**:
   - In **Eye Tracker 6000** open the menu and select Configure->Serial Out Port Settings. The dialog will open. Click on the button "Restore Defaults" and make sure that "Streaming Mode" checkbox is checked.
   - In **Eye Tracker 5000** open with Notepad file E5000.cfg located in the same folder with Eye Tracker (E5*.exe) and edit the line:
     ```
     serial_data_output_format_type=129
     ```

4. **Test the connection.** On the computer that runs 3rd party application run **ASL Serial Port Viewer** (*Start ->Programs->ASL Eye Tracker 6000->Serial Out Port Viewer*). Select *Standard Data Format, Streaming Mode,* set correct *Serial Port*. Figure below shows the Viewer settings. Verify that you are getting data.
   Hint: You can simulate gaze data by going to the Eye Tracker and selecting from the menu *Calibrate->Set Target Points*. Then move the mouse over the Scene POG window. The Eye Tracker will report mouse position in place of point of gaze data.

   If Serial Port Viewer does not read data stream:
   - Check the cable connection. Switch two serial cables (“Controller” and “Serial Out”)
   - Check serial port selection on the Viewer (COM1, COM2 etc.). It should match the computer port where cable is connected. Most desktops have only COM1 port, but some have COM2 etc.
   - Verify that Eye Tracker program is running. Repeat Step 3.



Data Format: [✓ Standard] From *.cfg file

Serial Port: COM2

Eye-Head Integration

Data format description (*.cfg): C:\Program Files\ASL Eye Tracker 6000\EyeTracking\E6000.cfg

[✓ Streaming Mode]

Eye Camera Update Rate (Hz): 60

Baud Rate: 57600

status	pupil_diam	spare_byte	horz_gaze_coord	vert_gaze_coord
0	60	0	594.8	1462
0	60	0	593.9	1467.1
0	60	0	596.3	1462.5
0	60	0	595.9	1468.9
0	60	0	598.3	1458.7
0	60	0	594.6	1469.5
0	60	0	597.7	1463
0	60	0	595.7	1471.4
0	60	0	595.9	1461.2
0	60	0	596.3	1469.7
0	60	0	597.4	1461.2
0	60	0	592.9	1475.2
0	60	0	596	1462.1
0	60	0	593.6	1469.1
0	60	0	593.9	1465.5
0	60	0	591.7	1469.3
0	60	0	592.7	1461.4
0	60	0	591.1	1470.7
0	60	0	596.3	1460.8
0	60	0	592	1473.2
0	60	0	594.2	1459.8
0	60	0	595.2	1468.4
0	60	0	594.5	1463.4
0	60	0	594.1	1471.1
0	60	0	597.5	1458.1
0	60	0	594	1473.6
0	60	0	597.6	1461.8
0	60	0	596.3	1471
0	60	0	596.6	1462.8

5. **Exit Serial Port Viewer** after the test.