ETVision Network Com

REAL-TIME NETWORK COMMUNICATION WITH EXTERNAL DEVICES

MANUAL VERSION 2.4
March 2024

ARGUS SCIENCE

techsupport@argusscience.com
Web site: www.argusscience.com

Table of Contents

1 Network communication with *ETVision*

Commands can be sent to the *ETVision* application from an external device via TCP/IP. Streaming Data and/or video can be received from *ETVision* by the external device via either TCP or UDP. Commands are used to send XDAT values to *ETVision*; to open, close and name data files on *ETVision*; to start and stop recording to data files on *ETVision*; and to initiate and stop streaming real-time data or video from *ETVision* to the external device.

If a command channel is open the *ETVision* IP address and port number are displayed on the *ETVision* "Network Configuration" dialog (*System Control Table*->"EyeData" tab->"Real-Time Input/Output" group->"Configuration" button).

A TCP "command socket" must be opened by the external device, and connected to *ETVision* to enable command communication. When a command channel is opened, command messages can be sent to *ETVision* by the external device, and in certain cases *ETVision* will send messages to the external device on this channel. Streaming data or video requires that a second TCP or UDP "data socket" be opened by the external device and connected to *ETVision.*

Supported commands and received data and video message formats are described in section 2, "*ETVision* Message Format".

To send commands from an external device

On *ETVision*:

- Click the "Listen" button on *System Control Table, "*EyeData" tab.

On the External device:

- Create a TCP Socket which will be referred to as the "command socket".
- connect the command socket with the *ETVision* IP address and port number.
- send one of the supported commands to the command socket.

To receive streaming UDP data or video from *ETVision*

On *ETVision*:

- "Send Data Result" and/or "Send Left Eye Video" or "Send Right Eye Video" or "Send Scene Video" must be checked on the *Network Configuration* dialog. (Note that only one video channel can be sent. If one of the video channels is selected digital data is also sent).

On the External device:

- If a command channel is not already opened, create a TCP "command" socket and connect it to the *ETVision* as previously described.
- Create a UDP socket for receiving data ("UDP data socket"), and a memory buffer in which to copy data that comes to the UDP socket receive buffer.

- Send the CMD_START_SDATA_UDP (or CMD_START_SVIDEO) message to the command socket.
- Data (or video) will begin streaming to the UDP data socket. Copy data from the UDP receive buffer to a memory buffer for processing. See "CMD_DATA_MSG(0x81)" and "CMD_DATA_JPG(0x82)" in section 2 for data record format description.
- Use the CMD_STOP_SDATA_UDP (or CMD_STOP_SVIDEO_UDP) command message to halt the data or video streaming.

To receive streaming TCP data or video from an *ETVision* data channel

On *ETVision*:

- "Send Data Result" and/or "Send Eye Video" or "Send Scene Video" must be checked on the *Network Configuration* dialog.

On the External device:

- If a command channel is not already opened, create a TCP "command" socket and connect it to the *ETVision* as previously described.
- Create a TCP socket for receiving data ("TCP data socket") and a memory buffer in which to copy data that comes to the TCP socket receive buffer.
- Send the CMD_SET_CONNECT_TYPE to the command socket with the proper argument for "send data" or "send video".
- Connect the TCP data socket with *ETVision* IP address and port number. Data (or video) will begin streaming as soon as the connection is made. Copy data from the TCP receive buffer to a memory buffer for processing. See "CMD_DATA_MSG(0x81)" and "CMD_DATA_JPG(0x82)" in section 2 for data record format description.
- Close the TCP data socket connection to halt data streaming.

To receive a single data item value from ETVision command channel

On the External device:

- If a command channel is not already opened, create a TCP "command socket" and connect it to the *ETVision* as previously described.
- Send CMD_GET_DATA_ITEM with data ID set to desired item (list of IDs is provided at the end of this section and again in section 2)
- Read the response on the command socket channel. Examine the response to first confirm that it is a non-error response to the CMD_GET_DATA_ITEM command, and then to get the data item value. The response format is specified in section 2 along with all variable IDs, and the corresponding variable types.

Remember to close all sockets when through.

All commands start with a 4 byte Argus Signature, and all include a "byte order" checksum. The Argus Signature is hex 20 41 47 53, which is the ASCII code for the characters: <space>AGS. For the Argus Signature and all other messages elements specified, the least significant byte is sent first; so the first byte sent will be hex53 (least significant byte of the Argus Signature) followed by hex47, etc.

To compute the checksum sum all of the bytes in the message (other than the checksum itself), ignore all but the least significant byte of the sum, and take the "twos compliment" negative of that byte. If all bytes in the resulting message are summed (including the checksum) the least significant byte of the sum should be zero.

For example, to set XDAT=100 (hex64) send the following 20 bytes in the order listed. The values below are hex values.

Byte 0: 53
Byte 1: 47
Byte 2: 41
Byte 3: 20
Byte 4: 14
Byte 5: 00
Byte 6: 00
Byte 7: 00
Byte 8: 05
Byte 9: 00
Byte 10: 00
Byte 11: 00
Byte 12: 83
Byte 13: 00
Byte 14: 00
Byte 15: 00
Byte 16: 64
Byte 17: 00
Byte 18: 00
Byte 19: 00

Bytes 0-3 are the ArgusSignature; bytes 4-7 are the message size; bytes 8-11 are the Set_XDAT command; bytes 12-15 are the checksum; and bytes 16-19 are the XDAT value to be set. To calculate the check sum first sum bytes 0 through 11 and bytes 16 through 19 yielding dec381=hex17D. Consider only the least significant byte (hex7D). Take the twos compliment negative of hex7D to get hex83.

To have an external device command the *ETVision* to record data, first use the CMD_SET_DATAFILE_NAME command to specify a file name, then the CMD_OPEN_DATAFILE to open a file with that name on the *ETVision* PC. Once a file is opened, CMD_START_DATAFILE_RECORDING and CMD_STOP_DATAFILE_RECORDING can be used to start and pause recording. Close the file with CMD_CLOSE_DATAFILE. Note that external control can be mixed with local *ETVision* control. For example, a data file can be opened manually on the *ETVision* application, and an external device can then command recording to start and stop, etc. A start recording command from an external device will have no affect if a file is not opened, a stop recording command will have no affect if recording has not started, and so forth.

The data message format, for real-time data sent from *ETVision*, is described in the "ETVision Message Format" section. See "CMD_DATA_MSG" and "CMD_DATA_JPG" at end of section 2. The message item called "CheckState" is an 8

byte (64 bit) value specifying the contents of the data buffer. Each bit represents one of the possible data items from the list of possible data items shown in the table, below. If the bit is set, the corresponding data item is present. Included data items will be in the data buffer in the same order that they appear in the table. The least significant bit of the 8 byte CheckState value (the least significant bit of byte 47 in the data message) is bit 0 and corresponds to the fist item listed in the table. Note that there are only 57 data items, so bits 0-56 are used, and bits 57-63 of CheckState will always be 0. The list of data items at the end of this section specifies the CheckState bit associated with each item.

The CheckState bit only indicates the presence or absence of a data item. The size of each item, if present, is that listed in the table at the end of this section. Some integer values have "scale factors", which are also listed in the table. To create a floating point value with the intended units of measure (inches or centimeters for distance values, degrees for angle values, and pixels for pupil diameter) first convert the value to a float, then multiply by the scale factor listed in the table. An explanation of each data item can be found in section 3 of this document.

All data sent by *ETVision* uses the "little endian" convention (least significant byte of each word is stored in smallest memory address).

Time stamp values on Data and Video messages represent hundreds of nanoseconds (E.e., a time stamp value of one would represent 100 nanoseconds).

Sample C#, VC++, ETNetLib, and MFC programs, using the *ETVision* network communication protocol, are provided as part of the *ETRemote* installation. After installing *ETRemote*, these samples can be found in *C:\Program Files\Argus Science \ETRemote\ET_SDK_Samples*.

Note: In the table, below, some ID bits indicate inclusion of the data item in the current row and also the data item in the following row.

CheckState bit	Data item	Type	Size (bytes)	Scale factor
0	start_of_record	Byte	1	1
1	status	Byte	1	1
2	overtime_count	Uint16	2	1
3	mark_value	Byte	1	1
4	XDAT	UInt16	2	1
5	CU_video_field_num	UInt16	2	1
6	left_pupil_pos_horz	UInt16	2	1
	right_pupil_pos_horz	UInt16	2	1
7	left_pupil_pos_vert	UInt16	2	1
	right_pupil_pos_vert	UInt16	2	1
8	left_pupil_diam	UInt16	2	0.01
	right_pupil_diam	UInt16	2	0.01
9	left_pupil_height	UInt16	2	0.01
	right_pupil_height	UInt16	2	0.01
10	left_cr_pos_horz	UInt16	2	1
	right_cr_pos_horz	UInt16	2	1
11	left_cr_pos_vert	UInt16	2	1
	right_cr_pos_vert	UInt16	2	1
12	left_cr_diam	UInt16	2	1
	right_cr_diam	UInt16	2	1
13	left_cr2_pos_horz	UInt16	2	1
	right_cr2_pos_horz	UInt16	2	1
14	left_cr2_pos_vert	UInt16	2	1
	right_cr2_pos_vert	UInt16	2	1
15	left_cr2_diam	UInt16	2	1
	right_cr2_diam	UInt16	2	1
16	horz_gaze_coord	Int16	2	0.1
17	vert_gaze_coord	Int16	2	0.1
18	horz_gaze_offset	Int16	2	1
19	vert_gaze_offset	Int16	2	1
20	vergence_angle	Single	4	1
21	verg_gaze_coord_x	Single	4	1
22	verg_gaze_coord_y	Single	4	1
23	verg_gaze_coord_z	Single	4	1
24	hdtrk_X	Int16	2	0.01
25	hdtrk_Y	Int16	2	0.01
26	hdtrk_Z	Int16	2	0.01
27	hdtrk_az	Int16	2	0.01
28	hdrtk_el	Int16	2	0.01
29	hdtrk_rl	Int16	2	0.01
30	ET3S_scene_number	Byte	1	1
31	ET3S _gaze_length	Single	4	1

32	ET3S _horz_gaze_coord	Single	4	1
33	ET3S _vert_gaze_coord	Single	4	1
34	SSC_horz_gaze_coord	Single	4	1
35	SSC_vert_gaze_coord	Single	4	1
36	left_eyelocation_X	Int16	2	0.01
	right_eyelocation_X			
37	left_eyelocation_Y	Int16	2	0.01
	right_eyelocation_Y			
38	left_eyelocation_Z	Int16	2	0.01
	right_eyelocation_Z			
39	left_gaze_dir_X	Int16	2	0.001
	right_gaze_dir_X			
40	left_gaze_dir_Y	Int16	2	0.001
	right_gaze_dir_Y			
41	left_gaze_dir_Z	Int16	2	0.001
	right_gaze_dir_Z			
42	aux_sensor_X	Int16	2	0.01
43	aux_sensor_Y	Int16	2	0.01
44	aux_sensor_Z	Int16	2	0.01
45	aux_sensor_az	Int16	2	0.01
46	aux_sensor_el	Int16	2	0.01
47	aux_sensor_rl	Int16	2	0.01
48	left_eyelid_upper_vert	Uint16	2	1
	right_eyelid_upper_vert	Uint16	2	1
49	left_eyelid_lower_vert	Uint16	2	1
	right_eyelid_lower_vert	Uint16	2	1
50	left_blink_confidence	Uint16	2	1
	right_blink_confidence	Uint16	2	1
51	left_ellipse_angle	single	4	1
	right_ellipse_angle	single	4	1
52	Gaze_LAOI	Uint32	4	1
53	LAOI_horz_gaze_coord	single	4	1
54	LAOI_vert_gaze_coord	single	4	1
55	fix_duration	single	4	1
56	horz_fix_coord	single	4	1
57	vert_fix_coord	single	4	1
58	Gaze_AI	Uint32	4	1

2 *ETVision* Message Format

Byte 0: Argus Signature // ArgusSignature (0x20414753): " AGS", 4 bytes
Byte 4: MsgSize // Total message size, 4 bytes
Byte 8: Cmd // Message Command, 1 byte, other bytes reserved to 0
Byte 12: Checksum // Message Checksum, 1 byte, other bytes reserved to 0
Byte 16: Argument // Optional, depends on command type

```
// Command Type
#define CMD_START_DATAFILE_RECORDING  1
#define CMD_STOP_DATAFILE_RECORDING   2
#define CMD_OPEN_DATAFILE             3
#define CMD_CLOSE_DATAFILE            4
#define CMD_SET_XDAT                  5
#define CMD_SET_DATAFILE_NAME         6
#define CMD_SET_CONNECT_TYPE          7
#define CMD_START_SDATA_UDP           8
#define CMD_STOP_SDATA_UDP            9
#define CMD_START_SVIDEO_UDP          10
#define CMD_STOP_SVIDEO_UDP           11
#define CMD_START_RVIDEO_UDP          12
#define CMD_STOP_RVIDEO_UDP           13
#define CMD_START_SVFILE_RECORDING    14
#define CMD_STOP_SVFILE_RECORDING     15
#define CMD_OPEN_SVFILE               16
#define CMD_CLOSE_SVFILE              17

// Special Command Type for EyeTracRemote
#define CMD_DISPLAY_TPS_FULLSCREEN    18
#define CMD_HIDE_TPS_FULLSCREEN       19
#define CMD_SET_TP_TOTALNUM           20
#define CMD_SHOW_TP                   21
#define CMD_HIDE_TP                   22

// Special Command Type with Response
#define CMD_GET_TP_TOTALNUM           23
#define CMD_GET_TP_POS                24
#define CMD_GET_DATAITEM              25

// Response bit for network command
#define CMD_RSP_BIT                   0x80000000
#define CMD_RSP_ERR_BIT               0x40000000

// Data / Video Command Type Message
#define CMD_DATA_MSG                  0x81
#define CMD_JPEG_MSG                  0x82
```

// Normal Command Type 1 - 17

1. **CMD_START_DATAFILE_RECORDING (1)**

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x00000001) // Message Command: Start Recording Data
Byte 12: Checksum (0x000000ef) // Message Checksum: checksum byte

2. **CMD_STOP_DATAFILE_RECORDING (2)**

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x00000002) // Message Command: Stop Recording Data
Byte 12: Checksum (0x000000ee) // Message Checksum: checksum byte

3. **CMD_OPEN_DATAFILE (3)**

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x00000003) // Message Command: Open Data File
Byte 12: Checksum (0x000000ed) // Message Checksum: checksum byte

4. **CMD_CLOSE_DATAFILE (4)**

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x00000004) // Message Command: Close Data File
Byte 12: Checksum (0x000000ec) // Message Checksum: checksum byte

5. **CMD_SET_XDAT (5)**

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000014) // Total message size: 20
Byte 8: Cmd (0x00000005) // Message Command: Set XDAT
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: Argument (XDAT) // XData Value

6. **CMD_SET_DATAFILE_NAME (6)**

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 16 + Size of Data File Name
Byte 8: Cmd (0x00000006) // Message Command: Set XDAT
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: Argument (Char String) // Data File Name Char String
Note: If file name already exists on specified *ETVision* folder, *ETVision* will not overwrite the previous file. Rather it will use the default file name instead.

7. **CMD_SET_CONNECT_TYPE (7)**

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000014) // Total message size: 20
Byte 8: Cmd (0x00000007) // Message Command: Set Connect Type
Byte 12: Checksum // Message Checksum: checksum byte

Byte 16: Argument // Connect Type

- #define SOCKET_TYPE_SDATA_TCP 3 // Send Data to Remote via TCP / IP
- #define SOCKET_TYPE_SVIDEO_TCP 7 // Send Video to Remote via TCP / IP
- #define SOCKET_TYPE_RVIDEO_TCP 9 // Receive Video from Remote via TCP / IP

8. CMD_START_SDATA_UDP (8)

Byte 0 ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000014) // Total message size: 20
Byte 8: Cmd (0x00000008) // Message Command: Start Sending UDP Data
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: Argument (Port Number) // UDP Port Number

9. CMD_STOP_SDATA_UDP (9)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x00000009) // Message Command: Stop Sending UDP Data
Byte 12: Checksum (0x000000e7) // Message Checksum: checksum byte

10. CMD_START_SVIDEO_UDP (10)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000014) // Total message size: 20
Byte 8: Cmd (0x0000000a) // Message Command: Start Sending UDP Video
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: Argument (Port Number) // UDP Port Number

11. CMD_STOP_SVIDEO_UDP (11)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x0000000b) // Message Command: Stop Sending UDP Video
Byte 12: Checksum (0x000000e5) // Message Checksum: checksum byte

12. CMD_START_RVIDEO_UDP (12)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000014) // Total message size: 20
Byte 8: Cmd (0x0000000c) // Message Command: Start Receiving UDP Video
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: Argument (Port Number) // UDP Port Number

13. CMD_STOP_RVIDEO_UDP (13)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x0000000d) // Message Command: Stop Receiving UDP Video
Byte 12: Checksum (0x000000e3) // Message Checksum: checksum byte

14. CMD_START_SVFILE_RECORDING (14)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16

Byte 8: Cmd (0x0000000e) // Message Command: Start Recording Screen Video
Byte 12: Checksum (0x000000e2) // Message Checksum: checksum byte

15. CMD_STOP_DATAFILE_RECORDING (15)
Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x0000000f) // Message Command: Stop Recording Screen Video
Byte 12: Checksum (0x000000e1) // Message Checksum: checksum byte

16. CMD_OPEN_SVFILE (16)
Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 16 + Size of Data File Name
Byte 8: Cmd (0x00000010) // Message Command: Open Screen Video File
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: Argument (Char String) // Screen Video File Name Char String

17. CMD_CLOSE_SVFILE (17)
Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x00000011) // Message Command: Close Screen Video File
Byte 12: Checksum (0x000000df) // Message Checksum: checksum byte

// 18 – 22 are Special Commands for communication with *EyeTracRemote* (these are commands sent by eyetracker to an open command socket and are <u>not</u> used by *ETVision*)

18. CMD_DISPLAY_TPS_FULLSCREEN (18)
Byte 0 ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x00000012) // Message Command: Display Target Points Full Screen
Byte 12: Checksum (0x000000de) // Message Checksum: checksum byte

19. CMD_HIDE_TPS_FULLSCREEN (19)
Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010) // Total message size: 16
Byte 8: Cmd (0x00000013) // Message Command: Hide Target Points Full Screen
Byte 12: Checksum (0x000000dd) // Message Checksum: checksum byte

20. CMD_SET_TP_TOTALNUM (20)
Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 20
Byte 8: Cmd (0x00000014) // Message Command: Set total number of Target Points
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: Argument (Total Number) // Total number of Target Points, maximum 20

21. CMD_SHOW_TP (21)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 28
Byte 8: Cmd (0x00000015) // Message Command: Display Target Point
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: TPIndex // Target Point Index, if TPIndex >= 20, show all points
Byte 20: TPH100 // Target Point Horizontal Coordinate x 100
Byte 24: TPV100 // Target Point Vertical Coordinate x 100

22. CMD_HIDE_TP (22)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 28
Byte 8: Cmd (0x00000016) // Message Command: Hide Target Point
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: TPIndex // Target Point Index, if TPIndex >= 20, hide all points
Byte 20: Reserved
Byte 24: Reserved

// 23 – 25 are Special Command Type with Response (23 & 24 are <u>not</u> applicable to *ETVision*)

23. CMD_GET_TP_TOTALNUM (23)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 16
Byte 8: Cmd (0x00000017) // Message Command: Get Total Target Points Number
Byte 12: Checksum // Message Checksum: checksum byte

// Response
Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 20
Byte 8: Response Cmd // Response Command:
// No Error: CMD_RSP_BIT | MD_GET_TP_TOTALNUM
// Error: CMD_RSP_BIT | CMD_RSP_ERR_BIT | MD_GET_TP_TOTALNUM
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: TotalNum // Total Target Points Number

24. CMD_GET_TP_POS (24)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 20
Byte 8: Cmd (0x00000018) // Message Command: Get Target Point Position
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: TPIndex // Target Point Index

// Response
Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"

Byte 4: MsgSize // Total message size: 28
Byte 8: Response Cmd // Response Command:
// No Error: CMD_RSP_BIT | MD_GET_TP_POS
// Error: CMD_RSP_BIT | CMD_RSP_ERR_BIT | MD_GET_TP_POS
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: TPIndex // Target Point Index
Byte 20: TPH100 // Target Point Horizontal Coordinate x 100
Byte 24: TPV100 // Target Point Vertical Coordinate x 100

25. CMD_GET_DATA_ITEM (25)

Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 20
Byte 8: Cmd (0x00000017) // Message Command: Get Eye Data Item Value
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: DataID // Data Item ID

// Response
Byte 0: ArgusSignature (0x20414753) // ArgusSignature: " AGS"
Byte 4: MsgSize // Total message size: 48
Byte 8: Response Cmd // Response Command:
// No Error: CMD_RSP_BIT | MD_GET_DATAITEM
// Error: CMD_RSP_BIT | CMD_RSP_ERR_BIT | MD_GET_DATAITEM
Byte 12: Checksum // Message Checksum: checksum byte
Byte 16: FrameNo // Current Frame Number
Byte 20: Reserved // Reserved bytes
Byte 24: TimeStamp // Time Stamp of this frame
Byte 32: UpdateRate // Frame Update Rate
Byte 36: DataID // Data Item ID
Byte 40: DataVal // Data Item Value
Byte 44: DataVal // 2nd value (for items with "2 Floating Point values)

```
// Data File Items defination (Note: items with "2 floating point values" are for left and right eye)
#define DATAID_START_OF_RECORD          0     // DataVal: Unsigned Integer Type
#define DATAID_STATUS                   1     // DataVal: Unsigned Integer Type
#define DATAID_OVERTIME_COUNT           2     // DataVal: Unsigned Integer Type
#define DATAID_MARK_VALUE               3     // DataVal: Unsigned Integer Type

#define DATAID_XDAT                     4     // DataVal: Unsigned Integer Type
#define DATAID_CU_VIDEO_FIELD_NUM       5     // DataVal: Unsigned Integer Type

#define DATAID_PUPIL_POS_HORZ           6     // DataVal: 2 Floating Point values
#define DATAID_PUPIL_POS_VERT           7     // DataVal: 2 Floating Point values

#define DATAID_PUPIL_DIAM               8     // DataVal: 2 Floating Point values
#define DATAID_PUPIL_HEIGHT             9     // DataVal: 2 Floating Point values

#define DATAID_CR_POS_HORZ              10    // DataVal: 2 Floating Point values
#define DATAID_CR_POS_VERT              11    // DataVal: 2 Floating Point values

#define DATAID_CR_DIAM                  12    // DataVal: 2 Floating Point values
```

```
#define DATAID_CR2_POS_HORZ          13    // DataVal: 2 Floating Point values
#define DATAID_CR2_POS_VERT          14    // DataVal: 2 Floating Point values

#define DATAID_CR2_DIAM              15    // DataVal: 2 Floating Point values

#define DATAID_HORZ_GAZE_COORD       16    // DataVal: Floating Point Type
#define DATAID_VERT_GAZE_COORD       17    // DataVal: Floating Point Type

#define DATAID_HORZ_GAZE_OFFSET      18    // DataVal: Floating Point Type
#define DATAID_VERT_GAZE_OFFSET      19    // DataVal: Floating Point Type

#define DATAID_VERGENCE_ANGLE        20    //Data Val: Floating Point Type
#define DATAID_VERG_GAZE_COORD_X     21    //Data Val: Floating Point Type
#define DATAID_VERG_GAZE_COORD_Y     22    //Data Val: Floating Point Type
#define DATAID_VERG_GAZE_COORD_Z     23    //Data Val: Floating Point Type

#define DATAID_HDTRK_X               24    // DataVal: Floating Point Type
#define DATAID_HDTRK_Y               25    // DataVal: Floating Point Type
#define DATAID_HDTRK_Z               26    // DataVal: Floating Point Type

#define DATAID_HDTRK_AZ              27    // DataVal: Floating Point Type
#define DATAID_HDTRK_EL              28    // DataVal: Floating Point Type
#define DATAID_HDTRK_RL              29    // DataVal: Floating Point Type

#define DATAID_ET3S_SCENE_NUMBER     30    // DataVal: Integer Type
#define DATAID_ ET3S _GAZE_LENGTH    31    // DataVal: Floating Point Type

#define DATAID_ ET3S _HORZ_GAZE_COORD 32   // DataVal: Floating Point Type
#define DATAID_ ET3S _VERT_GAZE_COORD 33   // DataVal: Floating Point Type

#define DATAID_SSC_HORZ_GAZE_COORD   34    // DataVal: Floating Point Type
#define DATAID_SSC_VERT_GAZE_COORD   35    // DataVal: Floating Point Type

#define DATAID_ EYE_LOCATION_X       36    // DataVal: 2 Floating Point values
#define DATAID_ EYE_LOCATION_Y       37    // DataVal: 2 Floating Point values
#define DATAID_ EYE_LOCATION_Z       38    // DataVal: 2 Floating Point values

#define DATAID_ GAZE_DIR_X           39    // DataVal: 2 Floating Point values
#define DATAID_ GAZE_DIR_Y           40    // DataVal: 2 Floating Point values
#define DATAID_ GAZE_DIR_Z           41    // DataVal: 2 Floating Point values

#define DATAID_AUX_SENSOR_X          42    // DataVal: Floating Point Type
#define DATAID_AUX_SENSOR_Y          43    // DataVal: Floating Point Type
#define DATAID_AUX_SENSOR_Z          44    // DataVal: Floating Point Type

#define DATAID_AUX_SENSOR_AZ         45    // DataVal: Floating Point Type
#define DATAID_AUX_SENSOR_EL         46    // DataVal: Floating Point Type
#define DATAID_AUX_SENSOR_RL         47    // DataVal: Floating Point Type

#define DATAID_EYELID_UPPER_VERT     48    //DataVal: 2 Floating Point values
#define DATAID_EYELID_LOWER_VERT     49    //DataVal: 2 Floating Point values
#define DATAID_BLINK_CONFIDENCE      50    //DataVal: 2 Floating Point values

#define DATAID_ELLIPSE_ANGLE         51    //DataVal: 2 Floating Point values
```

```
#define DATAID_GAZE_LAOI                52      //DataVal: Unsigned Integer Type
#define DATAID_LAOI_GAZE_HORZ_COORD     53      //DataVal: Floating Point Type
#define DATAID_LAOI_GAZE_VERT_COORD     54      //DataVal: Floating Point Type
#define DATAID_FIX_DURATION             55      //DataVal: Floating Point Type
#define DATAID_HORZ_FIX_COORD           56      //DataVal: Floating Point Type
#define DATAID_VERT_FIX_COORD           57      //DataVal: Floating Point Type
```

26. CMD_CLOSE_SVFILE (17)

```
Byte 0:  ArgusSignature (0x20414753)        // ArgusSignature: " AGS"
Byte 4: MsgSize (0x00000010)                // Total message size: 16
Byte 8: Cmd (0x00000011)                  // Message Command: Close Screen Video File
Byte 12: Checksum (0x000000df)              // Message Checksum: checksum byte
```

//format of streaming data records sent by *ETVison*

Data Message Format: CMD_DATA_MSG(0x81)

```
Byte 0:  ArgusSignature (0x20414753)        // ArgusSignature: " AGS"
Byte 4: MsgSize                             // Total message size: 56 + DataSize
Byte 8: Cmd (0x00000081)                    // Message Command: Data Message
Byte 12: Checksum (0x00000000)  // Message Checksum: Reserved to 0 for Data Message
Byte 16: DataSize                           // Selected Data Size
Byte 20: FrameSize(0x00000000)              // Video Frame Size: 0 since no video
Byte 24: FrameNo                            // Current Frame Number
Byte 28: Reserved                           // reserved bytes
Byte 32: TimeStamp                          // Time Stamp of this frame
Byte 40: UpdateRate                         // Frame Update Rate
Byte 44: Reserved                           // reserved bytes
Byte 48: CheckState                         // Data Selected State
Byte 56: BufStart                           // Data Buffer
```

//format of streaming video records sent by ETVison

// Note: *ETVision* does not include digital data with streaming Video frame records;

// therefore "DataSize", specified in Bytes 16-19, will always be zero.

Video Message Format: CMD_JPEG_MSG(0x82)

```
Byte 0:  ArgusSignature (0x20414753)        // ArgusSignature: " AGS"
Byte 4: MsgSize                  // Total message size: 56 + DataSize + FrameSize
Byte 8: Cmd (0x00000082)                    // Message Command: JPEG Message
Byte 12: Checksum (0x00000000)    // Message Checksum: Reserved to 0 for JPEG Message
Byte 16: DataSize                           // Selected Data Size
Byte 20: FrameSize                          // Video Frame Size (Encoded JPEG Image)
Byte 24: FrameNo                            // Current Frame Number
Byte 28: Reserved                           // reserved bytes
Byte 32: TimeStamp                          // Time Stamp of this frame
Byte 40: UpdateRate                         // Frame Update Rate
Byte 44: Reserved                           // reserved bytes
Byte 48: CheckState                         // Data Selected State
```

Byte 56: BufStart // Data Buffer
Byte 56 + DataSize: Buffer // Video Buffer

3 Data Item Explanation

Data values in streaming data records must be scaled by the factors listed in the table at the end of section 1 to represent the units described in the data item explanations. For example, to convert a gaze coordinate value to the scene camera pixel units described, first convert the integer value to a float and then multiply by 0.1.

Individual data items received using **CMD_GET_DATA_ITEM,** do not reaquire scaling, and are formatted as described under the **CMD_GET_DATA_ITEM** description in the previous section.

Start of record byte – fixed value 0xFA

Status byte – contains eye tracer status information.

Bit	Meaning (if 1)
0 (least significant)	Head tracker enabled, monocular system or left eye binocular
1	Head tracker enabled, right eye (binocular system only)
2	Cornea Reflection found, right eye (binocular system only)
3	Pupil Found, right eye (binocular system only)
4	Cornea Reflection found, monocular system or left eye binocular
5	Pupil Found, monocular system or left eye binocular
6	Right eye data was simulated for left/right eye data synchronization (binocular only)
7	Left eye data was simulated for left/right eye data synchronization (binocular only)

overtime count, 2 bytes, unsigned integer. Shows how many records were lost prior to this one. Typically contains the value zero.

Mark value byte – will be last integer "Mark" value entered by user.

XDAT – 16 bit integer set by external device.

CU video field number – Internal field (or record) number kept by system. It is the number of vertical sync pulses received from the eye camera since the *ETVision* program was activated, and rolls over to 0 after every 65535 fields. Useful mostly for debugging purposes.

pupil_pos – coordinates proportional to horizontal (0 to 640) and vertical (0 to 480) pupil position with respect to the eye camera field of view.

Pupil_diam – value proportional to diameter of the pupil image on the eye camera. More specifically, the value is major axis of the ellipse shape identified as the pupil image. Note that this value is computed to a fraction of a pixel. Note also that this value will not be affected by degree of image ellipticity, and will not change (except for measurement noise) if the diameter of the actual pupil does not change and camera to eye distance does not change. This pixel value is the value displayed on the *ETVision* Interface (on the Data Display Screen), and the value shown by the Argus Science data analysis program, *ETAnalysis*.

Pupil_height – the minor axis of the ellipse shape identified as the pupil image. Scaling is the same as that described above for pupil_diam. Note that unlike pupil diam (major ellipse axis) the minor axis of the ellipse image shape will change length as degree of ellipticity changes due to eye movement, even if true pupil diameter remains constant.

ellipse_angle – the angle of the major axes of the pupil ellipse (ellipse shape identified as the pupil image) with respect to the eye camera horizontal axis. Reported values range from + 90 degrees to – 89.99 degrees. (Note that +90 and –90 degrees defines the same ellipse orientation).

cr_pos -- coordinates proportional to horizontal (0 to 640) and vertical ()0 to 480) corneal reflection with respect to the eye camera field of view.

cr_diam – Diameter of the corneal reflection image in eye camera pixels.

gaze_coord – horizontal and vertical coordinates of computed point of gaze with respect to the head mounted scene camera 1280 x 720 pixel field of view (fov). Note that fractional pixel positions are represented. Also note that the values are signed. Negative values represent positions to the left, or above the scene camera fov, while values greater than 640 or 480 represent positions to the right of, or below the camera fov.

gaze_offset – Manual offset added to horizontal or vertical gaze coordinate in scene camera pixel units.

vergence angle – The angle, in degrees, between the left and right eye lines of gaze.

verg_gaze_coord – When ET3Space is **not** enabled, these are the X, Y, and Z coordinates for the computed point of gaze expressed in the scene camera coordinate frame. The scene camera coordinate frame has its origin at the scene camera lens aperture; with an x axis pointing along the camera optical axis; a y axis pointing to the subject's right, parallel to the horizontal camera pixel rows; and a z axis pointing down, parallel to the camera pixel columns. The coordinate values correspond to real distance values with units of either inches or centimeters depending on the "Position Units" selected on the *System Control Table*, *System Configuration* tab. When ET3Space **is** enabled, these are the X, Y, and Z coordinates specifying a unit vector (vector with a total length of 1) in the direction of gaze and expressed in the *ET3Space* global coordinate system. In other words, a line from the scene camera to the point of gaze should have the direction defined by this unit vector.

hdtrk – X, Y, Z position values and azimuth, elevation, and roll orientation values received by the *ETVision* system from a head tracker. Position values are in units of inches or centimeters (depending on which unit system was set in *ETVision* configuration dialog). Angles are in degrees.

ET3S_scene_number – number of scene plane first intersected by line of gaze, as computed by the *ETVision*, *ET3Space* feature. . If gaze is not detected to be within the boundaries of any defined scene plane, ET3S_scene_number is "-1".

ET3S_gaze_length – Distance from the eye to the point of gaze on the scene plane designated by ET3S_scene_number, as computed by the *ETVision, ET3Space* feature. The value is recorded or transmitted as a single precision floating point value with units of either inches or centimeters (depending on which unit system was set in *ETVision* configuration dialog).

ET3S_gaze_coord – The point of gaze in scene plane coordinates, on the scene plane designated by EH_scene_number, as computed by the *ETVision, ET3Space* feature. Coordinates are with respect to the coordinate frame defined on the scene plane by the *ET3Space* environment specifications. The "horizontal" value is the Y scene plane coordinate, and the "vertical" value is the Z coordinate. The value is recorded or transmitted as a single precision floating point value with units of either inches or centimeters (depending on which unit system was set in *ETVision* configuration dialog). If ET3S_scene_number is "-1" (gaze not within any defined scene plane boundary), then the reported coordinates are gaze intersection with an infinite extension of scene plane 0.

SSC_gaze_coord – The horizontal and vertical pixel coordinate of point-of-gaze on the "Stationary Scene Camera" image. These values are meaningful only if the Stationary Scene Camera feature has been enabled and set up as described in the *ET3Space* manual.

eye_location – The location in space, with respect to the *ET3Space* "global coordinate system, of the subject's eye. This location is computed, by the *ET3Space* feature, based on data received from the head tracker, and knowledge of the location of the eye with respect to the head tracker sensor. It does not depend on eye pointing direction. The units are inches or centimeters.

gaze_dir – A 3 dimensional unit vector (vector with a total length of 1) in the direction of gaze, represented with respect to the *ET3Space* "global coordinate system". These are dimensionless quantities that specify a direction and have no units.

aux_sensor – Not available with most head trackers. Consult Argus Science.

eyelid_lower_vert – best estimate of the vertical position of the lower eyelid boundary, directly below the pupil center. The position is reported with respect to the eye camera field of view, in camera pixel units. The range is from 1 at the top of the camera field of view to 239 at the bottom. A value of 0 means that no position was detected. If the bottom boundary of the iris is exposed, the system may sometimes report the position of the bottom of the iris instead of the lower eyelid. In these cases the lower eyelid and iris bottom are usually very close to each other.

eyelid_upper vert – best estimate of the vertical position of the upper eyelid boundary, directly above the pupil center. The position is reported with respect to the eye camera field of view, in camera pixel units. The range is from 1 at the top of the camera field of view to 239 at the bottom. A value of 0 means that no position was detected. If the top boundary of the iris is exposed, the system may sometimes report the position of the top of the iris instead of the upper eyelid. In these cases the upper eye lid and iris top are usually very close to each other.

blink_confidence – Estimated probability that an eye blink is in process. The range is 0 – 100. 0 indicates that either the system is very sure that the eyelids are opened and the pupil is exposed, or the pupil is undetected for some reason other than a blink. Higher values indicate increasing probability that a blink is in progress. Note that if the pupil remains undetected for a time that is much longer than a typical blink, the blink confidence will change to a low value, indicating that something other than a blink must be preventing pupil recognition.

Gaze_LAOI -- 32-bit integer specifying which currently defined Live Areas of Interest (LAOIs), if any, contain the current point of gaze. If no LAOIs are defined this value will always be zero. Currently defined LAOIs are always numbered in the order that they appear on the "List" drop down menu, on the *LAOI Configuration* dialog. The first LAOI at the top of the list is 0, and next is 1, the next is 2, and so on. Each data record includes an integer value with each bit corresponding to one of the currently defined LAOIs. Bit 0 (the least significant bit) corresponds to LAOI 0, bit 1 corresponds to LAOI 1, etc. If gaze is not detected to be within an LAOI, the corresponding bit will be 0 on that data record. If gaze is detected to be within an LAOI, the corresponding bit will be 1. Note that if Live Areas of Interest overlap, gaze can simultaneously be in more than one, and in this case bits corresponding to all LAOIs containing the point of gaze will be set.

LAOI_gaze_coordinate – single precision floating point values cooresponding to horizontal and vertical gaze coordinates with respect to a "Live Area of Interest" (LAOI) coordinate system defined by the user. These coordinates are reported only when gaze is detected to be within the LAOI boundary. If gaze is not within the boundary of an LAOI or is in an LAOI with no defined coordinate system, the values will be "0.0". If gaze is within multiple overlapping LAOIs, LAOI gaze coordinates are reported for the lowest numbered LAOI for which a coordinate system has been defined.

fix_duration – single precision floating point value representing the current duration, in seconds (to the nearest 0.001 sec), of an on-going fixation, as determined by the real-time fixation detection algorithm (see ETVision manual, section **Error! Reference source not found.**). If there is not an "on-going" fixation, the value is 0.0.

fix_coord -- single precision floating point value representing the horizontal and vertical coordinates of an "on-going" fixation with respect to the head mounted scene camera 1280 x 720 pixel field-of-view (fov), as determined by the real-time fixation detection algorithm (see ETVision manual, section **Error! Reference source not found.**). If there is not an "on-going" fixation, the values are 0.0.

Gaze_AI – 32 bit Integer value specifying the type of AI object detection bounding box that contains the point-of-gaze cursor. The integer value is the index number of an object in the AI object detection model being used by *ETVision.* If gaze is not within any AI object bounding box, Gaze_AI is 0. If "AI Auto Object Detection" is not enabled, Gaze_AI is always 0. Note that if AI object bounding boxes overlap gaze can be inside more than one. In this case the system determines which bounding box center is closest to the gaze point and Gaze_AI reports the index for that object type.

4 Sample Programs

Sample programs are provided to demonstrate communication with *ETVision* from C++, C#, or Python applications. The examples include source code where appropriate. One set of C++ samples, and the Python examples demonstrate very basic communicate with *ETVision*, using just the Command Prompt window rather than a Graphic User Interface (GUI). C# and C++/MFC samples demonstrate slightly more sophisticated communication and include a GUI, and use a multi-threading library provided by Argus.

A ".Net" Com library, called ETNetLib.dll, is provided and the C# and C++/MFC samples use this library to implement communication. ETNetLib uses multi-threading, and the users application must support multi-threading if the library is used. All samples are normally installed as part of the *ETRemote* installation. On Win 10 PCs they are found from the Start symbol in *Start->Argus Science->ET SDK Samples*, or from Windows explorer in *Program Files (86)\Argus Science \ETRemote\ET_SDK_Samples*.

4.1 ETNetLib.DLL

ETNetLib is a COM library that provides an interface to *ETVision*. It can be used to send commands to *ETVision* and to receive streaming data or video using a separate *ETVision* data socket channel. It uses multi-threading and an application must support multi-threading to use the library.

For example, use "ConnectET" to open both an *ETVision* command and data socket. The library will initiate and maintain a thread to receive and buffer streaming data from *ETVision.* Use "GetConnectStatus" to ensure that valid connections are open. Use "GetDataItemValue" to get the latest available sample of any data value being streamed from *ETVision.* The value returned will be properly scaled and returned as the appropriate data type (unsigned integer, integer, or double). Use "SendETCmd" to send commands to *ETVision* via the command channel.

See the sample C# and C++/MFC programs for detailed examples. Both the compiled ETNetLib.dll and the source code, with a complete Microsoft Visual Studio project, are provided.

4.2 C++

The C++ examples are executed by typing commands on the "Command Prompt" window (Win 10: Start->All programs->Windows System->Command Prompt). Results also display on the Command Prompt window. Source code and Visual Studio project files are included for all samples. Be sure that the external PC and *ETVision* PC are connected to the same LAN. In all cases, if a command socket connection with *ETVision* is not already open, be sure to click "Listen" on the *ETVision* Network configuration dialog before running the sample program. All sample programs include a "readme" text file with instructions. The following sample programs are included. Note that these C++ samples do not use either the MFC library or the ETNetLIb COM Object library

- **ETGetDataItemValue**
  Demonstrate receiving ETVision data values on the command channel
  Demonstration of receiving single *ETVision* data values on the command channel. Be sure that "Send Data Result" is checked on the *ETVision* Network Configuration dialog. The command socket connection to *ETVision* is used to return a single item value. The *ETVision* IP address and port number as well as the desired data item are specified as command line arguments.

  Command: *ETGetDataItemValue IPAddress IPPort DataID*
  where IPAddress and IPPort are the address and port number shown on the *ETVision* Network configuration dialog, and DataID is the ID number of the desired data item (see section 2, command 25).

- **ETReceiveData**
  Demonstrate receiving streaming data from ETVision on a data channel
  Be sure that "Send Data Result" is checked on the *ETVision* Network Configuration dialog. Both a command and a data socket connection to *ETVision* are opened, and the data socket is used to receive streaming data. The *ETVision* IP address and port number are specified in the command line. A data record, containing all data items being reported by *ETVision*, is displayed as a line of values on the Command Prompt window. This will repeat for successive data records until the <Esc> key is pressed.

  Command: *ETReceiveData IPAddress IPPort*
  where IPAddress and IPPort are the address and port number shown on the *ETVision* Network configuration dialog

- **ETReceiveImage**
  Demonstrate receiving image data from ETVision on a data channel
  Be sure that one of the "Send Video" boxes is checked on the *ETVision, Network Configuration* dialog, and click "Listen". Both a command and a data socket connection to *ETVision* are opened by the ETReceiveImage program, and the data socket is used to receive image data. The *ETVision* IP address and port number as well as a jpg file name are specified in the command line. A single jpg image is read from *ETVision* and stored in the specified file. The command and data socket connections are then closed.

  Command: *ETReceiveImage IPAddress IPPort ImageFileName*
  where IPAddress and IPPort are the address and port number shown on the *ETVision* Network configuration dialog and ImageFileName is the name under which a jpg image file will be stored.

- **ETRecordData**
  Demonstrate control of ETVision data files
  This sample program sends commands to *ETVision* to name and open data files, and to start and stop recording data to the file. The *ETVision* IP address and port number as well as a data file name are specified in the command line. An *ETVision* command socket is opened. A command is sent to *ETVision* to open a file with the specified,

and then and a command is sent to begin recording. The *ETVision* Data File tab, on the System Control Table window, should show that a file has been opened and is recording. When the <Esc> key is pressed, on the PC running the ETRecordData sample program, a command will be sent to *ETVision* to Stop recording, close the data file, and close the command connection.

Command: *ETRecordData IPAddress IPPort FileName*
where IPAddress and IPPort are the address and port number shown on the *ETVision* Network configuration dialog and IPPortFileName is the name of the data file to be opened on *ETVision.*

- **ETSetXDAT**
  ***Demonstrate setting XDAT values on** ETVision*
  The *ETVision* IP address and port number as well as the desired XDAT value are specified in the command line. An *ETVision* command socket is opened, and a command is sent to set the specified XDAT value. The XDAT value should appear on the *ETVision* Data Display Screen window.

  Command: *ETRecordData IPAddress IPPort XDAT*
  where IPAddress and IPPort are the address and port number shown on the *ETVision* Network configuration dialog, and XDAT is an integer between 0 and 65535. The command socket connection is then closed.

4.3 C#

The C# sample programs use the ETNetLib COM Object library and demonstrate receiving realtime data, controlling *ETVision* data files, and sending XDAT values to *ETVision.* ETNetLib must be "registered" on the PC running the sample C# programs. If *ETRemote* was installed on the PC, this will have been done by the install program. Otherwise use the regsvr32 command to register the dll.

These samples include simple GUIs rather than requiring use of the Command Prompt window. Source code and Visual Studio project files are included for all samples. Be sure that the external PC and *ETVision* PC are connected to the same LAN. In all cases, if a command socket connection with *ETVision* is not already open, be sure to click "Listen" on the Network configuration dialog before running the sample program. All sample programs include a "readme" text file with instructions. The following sample programs are included.

- **ETReceiveData**
  Demonstrate receiving data on a streaming data channel
  Be sure that "Send Data Result" is checked on the *ETVision* Network Configuration dialog. To run the program, double click ETReceiveData.exe in the usual way. On the resulting program window, enter the *ETVision* IP address and port number shown on the *ETVision* Network configuration dialog. Click the "Connect" button to establish a connection to *ETVision*. The program will attempt to open both a command and data socket connection to *ETVision*. If successful, the Connect button will change to a "Disconnect" button. Click the "Receive Data" button to display the most recent data frame being sent over the

data channel. Click again to display another frame. Examine the ETReceiveDataForm.cs file code to see the calls to ETNetLib functions.

- **ETReceiveImage**
  Demonstrate receiving Image data on a streaming data channel
  Be sure that one of the Send Video boxes is checked on the *ETVision*, Network Configuration dialog, and click "Listen". On the PC running ETRecieveImage, double click "ETReceiveImage.exe" in the usual way. On the resulting program window, enter the *ETVision* IP address and port number shown on the *ETVision* Network configuration dialog. Click the "Connect" button to establish a connection to *ETVision*. The program will attempt to open both a command and data socket connection to *ETVision*. If successful, the Connect button will change to a "Disconnect" button. Click the "Receive Image" button to display the latest video frame being sent over the data channel. Look at the ETReceiveImageForm.cs file code to see the calls to ETNetLib functions.

- **ETRecordData**
  Demonstrate control of ETVision data files using the ETVision command socket channel
  To run the program, double click ETReceiveImage.exe in the usual way. On the resulting program window, enter the *ETVision* IP address and port number shown on the *ETVision* Network Configuration dialog. Click the "Connect" button to establish a command socket connection to *ETVision*. If successful, the Connect button will change to a "Disconnect" button. Type in a file name to be used as the name of the *ETVision* data file, then click "Set File Name". The file name should appear as the default name on the *ETVision* Data File dialog (Data File tab, on *ETVision* System Control Table). Click the "Open Host File" button and the file should appear as opened on the *ETVision* Data File dialog. Use the "Start Recording" and "Stop Recording" buttons to cause recording to start and stop on *ETVision*. This should be properly indicated on the *ETVision* Data File dialog. Examine the ETRecordDataForm.cs file code to see the calls to ETNetLib functions.

- **ETSetXDAT**
  Demonstrate setting ETVsion XDAT values using the ETVision command socket channel
  To run the program, double click ETSetXDAT.exe in the usual way. On the resulting program window, enter the *ETVision* IP address and port number shown on the *ETVision* Network configuration dialog. Click the "Connect" button to establish a command socket connection to *ETVision*. If successful, the Connect button will change to a "Disconnect" button. Type in the desired XDAT value and click the "Set" button. The XDAT value should appear on the *ETVision* Data Display Screen. Examine the ETSetXDATForm.cs file code to see the calls to ETNetLib functions.

4.4 C++/MFC

A set of 4 sample programs are provided using C++ and using the Micorsoft Foundation Class (MFC) library to generate a GUI. These are identical in program name and functionality to the C# samples, and also use the ETNetLib COM Object library.

4.5 Python

The Python examples are executed by typing commands on the "Command Prompt" window (Win 10: Start->All programs->Windows System->Command Prompt). Results also display on the Command Prompt window. Python code for each sample is provided as a "py" file. Python is an interpretive language, so separate executables are not provided. If Python with its associated libraries has been properly installed, the "py" files can be run from a command prompt window.

Be sure that the external PC and *ETVision* PC are connected to the same LAN. In all cases, if a command socket connection with *ETVision* is not already open, be sure to click "Listen" on the *ETVision* Network configuration dialog before running the sample program. All sample programs include a "readme" text file with instructions. The following sample programs are included. Note that the Python samples do not require either the MFC library or the ETNetLIb COM Object library

- **ETReceiveData**
  Demonstrate receiving streaming data from ETVision on a data channel
  Be sure that "Send Data Result" is checked on the *ETVision* Network Configuration dialog. Both a command and a data socket connection to *ETVision* are opened, and the data socket is used to receive streaming data. The *ETVision* IP address and port number are specified in the command line. A data record, containing all data items being reported by *ETVision*, is displayed as a line of values on the Command Prompt window. This will repeat for successive data records until the <Esc> key is pressed.

  Command: *py ETReceiveData.py IPAddress IPPort*
  where *IPAddress* and *IPPort* are the address and port number shown on the *ETVision* Network configuration dialog

- **ETRecordData**
  Demonstrate control of ETVision data files
  This sample program sends commands to *ETVision* to name and open data files, and to start and stop recording data to the file. The *ETVision* IP address and port number as well as a data file name are specified in the command line. An *ETVision* command socket is opened. A command is sent to *ETVision* to open a file with the specified, and then and a command is sent to begin recording. The *ETVision* Data File tab, on the System Control Table window, should show that a file has been opened and is recording. When the <Esc> key is pressed, on the PC running the *ETRecordData.py* sample program, a command will be sent to *ETVision* to Stop recording, close the data file, and close the command connection.

  Command: *py ETRecordData.py IPAddress IPPort FileName*

where *IPAddress* and *IPPort* are the address and port number shown on the *ETVision* Network configuration dialog and *FileName* is the name of the data file to be opened on *ETVision*.

- **ETSetXDAT**
  ***Demonstrate setting XDAT values on** ETVision*
  The *ETVision* IP address and port number as well as the desired XDAT value are specified in the command line. An *ETVision* command socket is opened, and a command is sent to set the specified XDAT value. The XDAT value should appear on the *ETVision* Data Display Screen window.

  Command: *py ETRecordData.py IPAddress IPPort XDAT*
  where *IPAddress* and *IPPort* are the address and port number shown on the *ETVision* Network configuration dialog, and *XDAT* is an integer between 0 and 65535. The command socket connection is then closed.

- **ETReceiveImage**
  Demonstrate receiving Image data on a streaming data channel
  This sample program opens a command socket and data socket to *ETVision*, sends a command to *ETVision* to stream video, receives one video frame from the data socket and records it as a jpg image to the specified file. Be sure that one of the Send Video boxes is checked on the *ETVision*, Network Configuration dialog. The *ETVision* IP address and port number (shown on the *ETVision* Network configuration dialog) as well as the image file name are specified in the Python command line. From a command prompt, go to the ETReceiveImage.py folder and use the command line shown below.

- Command: *py ETReceiveImage.py IPAddress IPPort ImageFileName*
  where *IPAddress* and *IPPort* are the address and port number shown on the *ETVision* Network configuration dialog.

5 Communication with *Paradigm*

Paradigm with the *Paradigm elements for ASL* add-on package, from Perception Research Sytems Inc., is an application designed to facilitate building precisely timed visual display experiments and collecting subject responses. Experiments can be built with a drag and drop designer. A simple script event allows insertion of Python script if needed. The *Paradigm elements for ASL* add-on package provides drag and drop elements to communicate with an Argus ETVision eye tracker, including control of data recording on an *ETVision* eyetracker, and setting XDAT values on *ETVision*.

To design an experiment that will communicate with *ETVision*, first open a new experiment. Under Devices, select the Device Manager, and highlight "Network Port". Check the "NetworkPort1" check box. Next to "IP Address", enter the IP address shown on the *ETVision* Network Configuration dialog. Click on the Experiment Bar to open the network port. Select the "ASL" tab on the left pane of the Paradigm window, and drag the "Begin Session" element to the "Commands" pane near the bottom right of the Paradigm window. A small dialog window will appear with a drop down menu labeled "XDAT Port Type". Select "EyeTRAC7 port" from the drop down menu, and click "Done".

Follow instructions in the Paradigm documentation to design an experiment using drag and drop events and other Paradigm features. For example, command *ETVision* to start recording at any event by dragging the ASL tab "Start Recording" element to the Command pane. Set an XDAT value during any event by dragging the ASL tab "Send Marker" element to the Command pane and entering the desired XDAT value. See Paradigm documentation for additional features and more detailed instructions.

6 Communication with *E-Prime*

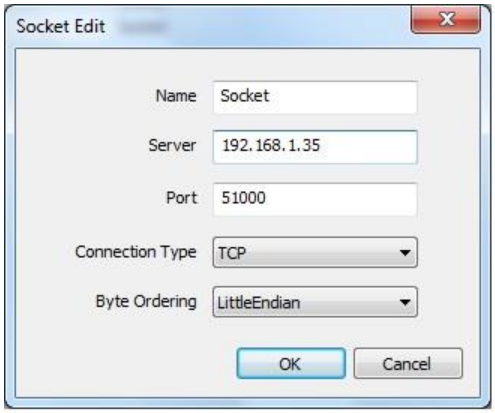
E-Prime, from Psychology Software Tools, Inc., is an application for designing visual display experiments. It includes a programming language called E-Basic as well as graphical drag and drop features. *E-Prime* version 2.0 Professional supports network TCIP sockets and can communicate with the *ETVision* eyetracker. Argus Science provides an E-Basic library script with subroutines that can be called from *E-Prime* "in-line" script objects. The library routines support control of *ETVision* data recording functions, sending "XDAT" values to *ETVision*, and reading real-time gaze data from *ETVision*. Sample scripts are provided to demonstrate control of *ETVision* data files, setting XDAT values, and reading real time gaze values. The sample scripts have been tested with E-Prime 2.0 (SP2) (2.0.10.356).

The library script and samples are normally installed as part of the *ETRemote* installation. On Win 10 PCs click *Start->Argus Science->ET SDK Samples*, or use windows explorer to navigate to *Program Files (86)\Argus Science \ETRemote\ET_SDK_Samples*. A windows explorer screen will show several folders including "Eprime".

If *ETRemote* is being used to send stimulus screen image to the *ETVision* for display as a "Stationary Scene Camera" image on the *ETVision* Scene Video Screen, E-Prime may sometimes prevent *ETRemote* from grabbing a screen image. In this case the image can always be made to display on *ETVision* by inserting a zero duration blank image just before the image object in question. This is illustrated by one of the included sample scripts.

When starting a new experiment design on E-Studio, double click the "Experiment Object" node on the "Structure" diagram to bring up the Properties dialog. Select the Devices tab. If "Socket" is listed, be sure it is checked. If "Socket" is not listed as one of the devices, click the "Add" button, highlight "Socket" and click OK. Double click "Socket" on the Properties, Devices tab, to bring up the Socket Edit dialog.

On the socket edit dialog, set "Name" to "Socket"; set "Server:" to the IP address shown on the *ETVision* Network Configuration dialog; and set "Port", "Connection Type", and "Byte Ordering" as shown in the screen shot below.



Open "Argus_ETVision_Eprime_Library.txt" in Notepad, and use copy/paste to past the text into the E-Studio User Script Window. The library subroutines can now be called from E-Prime "In-line Objects".

The library subroutines as well as the sample programs that use them are designed to be very short and simple. A user with experience in E-Basic programming can customize these routines to best suit their specific task.

Functionality of each "library" subroutine is described by comment lines included with the code. The Subroutines are:

- ETV_OpenDataFile
- ETV_CloseDataFile
- ETV_StartDataFileRecording
- ETV_StopDataFileRecording
- SendXDATtoETV (XDAT As Long)
- ETV_GetRealTimeFloatData (DataID As Integer, Value As Single, status As Integer)
- ETV_GetRealTimeIntegerData (DataID As Integer, Value As Integer, status As Integer)

To run one of the sample scripts, select "Open" from the E-Studio File menu, browse to the desired *.es2 file, and click Open. It will not be necessary to copy text from the Argus_ETVision_Eprime_Library, since that will already be part of the sample script. It will, however, be necessary to open the Socket Edit menu, and set "Server" to the *ETVision* IP address, as previously described. Make sure other items on the Socket Edit dialog are also set as previously shown. If a command channel is not already opened on *ETVision*, click the "Listen" button on the *ETVision* Network Configuration dialog before running the script. The following sample scripts are included:

ETVision_Sample_FileOpen_Rec_SendXDAT_FileClose.es2

The sample program will first set XDAT to 0 (call to "SendXDATtoETV" subroutine), send a command to *ETVision* to open a file (call to "ETV_OpenDataFile"), and will display a message explaining the demonstration and prompting for <SpaceBar> to start. The opened data file should show as ready to record on the *ETVision* Data File dialog.

When <SpaceBar> is pressed E-Prime will send a command to *ETVision* to Start recording on the data file (call to "ETV_StartDataFileRecording"), and the *ETVision* Data File dialog should show that the file is recording.

E-Prime will then sequentially display a set of 6 screens, each with a different background color and a number in the center. Each time a new screen is displayed XDAT will be set to the number shown on the screen. This should be visible on the *ETVision* Data Display screen. Each screen will display for several seconds before proceeding to the next display. After all 6 have been displayed, XDAT will be set back to zero ("SendXDATtoETV"), the file will stop recording ("ETV_StopDataFileRecording"), and will close ("ETV_CloseDataFile").

ETVision_Sample_ReadRTData.es2

The sample program will first display an instruction screen. When <SpaceBar> is pressed, E-Prime will loop though *ETVision* commands to send horizontal and vertical gaze data over the command channel ("ETV_GetRealTimeFloatData" subroutine), and to update the E-Prime display with these values.

ETVision_BasicRT.es2

If *ETRemote* is being used to send stimulus screen image to the *ETVision* for display as a "Stationary Scene Camera" image on the *ETVision* Scene Video Screen, E-Prime may sometimes prevent *ETRemote* from grabbing a screen image. In this case the image will appear on the Stimulus PC screen, but will not show on the *ETVision* Scene Video Screen. The image object can be made to appear by inserting a blank text object with zero duration just before it.

E-Prime includes a sample reaction time experiment called BasicRT.es2. ETVision_BasicRT.es2 is a modified version. It differs from the original only in that there are three additional text objects labeled Blank1, Blank2, and Blank3, each with zero duration. If Blank1 is removed, the Instruction screen is not captured by *ETRemote*, and does not display on the *ETVision* Scene Video Screen. If Blank2 is removed some of the TrialProc text objects will not display on the *ETVision* Scene Video Screen, and if Blank3 is removed the Goodbye text does not display on the *ETVision* Scene Video Screen. With the "blanks" included, all images that display on the Stimulus screen also appear on the *ETVision* Scene Video Screen.

7 Communication with *MATLAB*

MATLAB, from The Mathworks, Inc., when also equipped with the "Instrument Control Toolbox", can open TCP sockets, and can communicate with Argus *ETVision* systems. Call the tcpip function, with a device IP address and port number as arguments, to create a tcpip object. The *ETVision* IP address and port number can be found on the *ETVision* Network Configuration dialog. For example, create the tcp object *t* with IP address 192.168.1.35 and port 51000, as follows:

t = tcpip_pc ('192.168.1.35', 51000)

Use fopen to connect. For example:

fopnen(t)

Remember to close the connection and delete the tcpip object at the end of the program. For example:

fclose(t)
delete(t)

A set of function subroutines is provided to send various commands to *ETVision*. Each is in a separate file, with a file name that is the same as the function, and ".m" extension. Each assumes that a tcpip object has already been created and connected with *ETVision* as the *ETVision* command socket. The tcpip object is the first argument in each function. There are separate functions to send commands for *ETVision* file open, file close, start and stop recording, set XDAT, and to command *ETVision* to send the latest value of a data item over the command socket. The latter two functions require a second argument, to specify the desired XDAT value in the first case, and to specify the DataID of the desired data item, in the second case. XDAT must be an integer between 0 and 65535. DataID must be an integer corresponding to one of the data ID values listed in section three, under command 25. The functions are:

- ETVision_OpenDataFile(t)
- ETVision_CloseDataFile(t)
- ETVision_StartDataFileRecording(t)
- ETVision_StopDataFileRecording(t)
- ETVision_SendXDAT(t, XDAT)
- ETVision_GetDataItem(t, DataID)

3 sample Matlab scripts are provided, which call the functions discussed above, so the files containing these functions must be in the current Matlab directory or a directory on the Matlab path. Each sample starts by creating a tcp object and connecting to *ETVision*, as discussed above. The Matlab PC and *ETVision* PC must be connected to the same local area network. When running the sample, the user must first edit the line of code that sets the proper IP address, since this may be different for each system installation.

Note: 32 bit versions of MATLAB can use the ArgusFileLib.dll if it has been registered. 64 bit versions of MATLAB will not be able to use this 32 bit dll.

ETVision_Sample_SendXDAT.m
The sample program creates a TCP object, opens a command socket connection with *ETVision*, pauses for 1 sec, sets XDAT to 500 on *ETVision* by calling the ETVision_SendXDAT function, pauses for another seconds, and then closes the connection and deletes the TCP object. The user must edit the script to include the correct IP address for *ETVision* (as shown on the *ETVision* Network Configuration dialog), and to change the XDAT value. Be sure that the "Listen" button has been clicked, on the *ETVision* Network configuration dialog, before running the script. The XDAT values set by the sample

ETVision_Sample_File_and_XDAT_Commands.m
The sample program creates a TCP object, opens a command socket connection with *ETVision*, pauses for 1 sec, then sends a command to open a data file on *ETVision*, a command to set XDAT=0, and a command to start recording on the opened file. At 5 sec intervals, commands are sent to set XDAT to 100, 150, and 200. After another 5 sec interval, commands are sent to close the connection and delete the TCP object.

ETVision_Sample_Read_RealTime_Data.m
The sample program creates a TCP object, opens a command socket connection with *ETVision*, and pauses for 1 sec. The program then clears the input buffer, and sends a command to *ETVision* to send the latest horizontal gaze position value over the command channel. It then reads the first 40 bytes of the response into an input buffer, and reads a float value starting at the 41st byte into a horizontal gaze position variable. The command and read sequence is repeated for vertical gaze and pupil diameter values. Finally the input buffer is cleared and commands are sent to close the connection and delete the TCP object. Note that the sample program does not check the *ETVision* response to see if the Argus signature is correct, to see if the data ID on the return value is correct, or to see if the checksum on the return value is correct. This type of error checking can be included if desired. As with the other sample scripts, be sure that the "Listen" button has been clicked, on the *ETVision* Network configuration dialog, before running the script.

8 Communication using Lab Streaming Layer

"Lab Streaming Layer" (LSL) is an open source protocol that allows researchers to receive and synchronize streaming data from multiple devices.

Sample Code description
Argus Science provides sample CPP source code, for a program called *ETRelayData*, to illustrate a procedure for receiving an *ETVision* data stream via LAN and streaming the same data to LAN with an LSL compatible protocol. Since LSL protocol requires that there be a known number of data streams, the sample code assumes that the data stream made available to LSL contains the default data item set output from *ETVision* when not using the *ET3Space* feature. Users can use the sample code as is, can modify the code to change the set of data items made available to LSL, or may just create and embed equivalent code in their own applications.

Sample CPP code, for a program called *ETReceiveData*, is also provided to illustrate a procedure for receiving the LSL compatible data stream output by *ETRelayData*. Users will usually want to use the *ETReceiveData* code as an example to help them embed equivalent code in their own application, since the *ETReceiveData* sample illustrates how to receive the data stream but does not do anything with it.

In both cases (*ETRelayData* and *ETReceiveData*) CPP source code and a *Visual Studio 2022* project file are provided. An executable is not included since it is assumed that users will want to customize the data set for their applications. Users can use the provided project file with Windows 10 or Windows 11, and Visual Studio 2022; or can write comparable code with whatever development environment is being used.

If *ETVision* is not set to the "non-ET3Space" default data set, any data items in the default set not actually being streamed by *ETVision* will still be included in the LSL compatible output stream produced by *ETRelay Data*, but will always have values of zero. Any data items output by *ETVision* that are not part of the default set will simply not appear in the LSL compatible output stream. The default data item set includes the following items:

```
"Frame Number",
"Update Rate",
"overtime_count",
"mark_value",
"XDAT",
"left_pupil_pos_horz",
"right_pupil_pos_horz",
"left_pupil_pos_vert",
"right_pupil_pos_vert",
"left_pupil_diam",
"right_pupil_diam",
"left_cr_pos_horz",
"right_cr_pos_horz",
"left_cr_pos_vert",
"right_cr_pos_vert",
```

```
      "left_cr2_pos_horz",
      "right_cr2_pos_horz",
      "left_cr2_pos_vert",
      "right_cr2_pos_vert",
      "horz_gaze_coord",
      "vert_gaze_coord",
      "vergence_angle",
      "verg_gaze_coord_x",
      "verg_gaze_coord_y",
      "verg_gaze_coord_z"
```

Explanations for these items can be found in section 3 of this document, as well as in the *ETVision* manual.

Modifying the *ETRelayData* code for different data sets

The provided sample code can be easily modified to relay a different data set. For example, to delete the last three items ("verg_gaze_coord_x", verg_gaze_coord_y" and "verg_gaze_cord_z") from the data set, modify the code as follows.

1. On "ETRelayData.h", reduce the `NUM_ARGUS_LSL_CHANNELS` by 3. In other words, change "`#define NUM_ARGUS_LSL_CHANNELS  25`" to "`#define NUM_ARGUS_LSL_CHANNELS  22`".

2. On "ETRelayData.cpp" delete the last three items listed under the comment "LSL Streaming Channels". The list will now be:

```
const char *channels[] = {
      "Frame Number",
      "Update Rate",
      "overtime_count",
      "mark_value",
      "XDAT",
      "left_pupil_pos_horz",
      "right_pupil_pos_horz",
      "left_pupil_pos_vert",
      "right_pupil_pos_vert",
      "left_pupil_diam",
      "right_pupil_diam",
      "left_cr_pos_horz",
      "right_cr_pos_horz",
      "left_cr_pos_vert",
      "right_cr_pos_vert",
      "left_cr2_pos_horz",
      "right_cr2_pos_horz",
      "left_cr2_pos_vert",
      "right_cr2_pos_vert",
      "horz_gaze_coord",
      "vert_gaze_coord",
      "vergence_angle"
};
```

3. On "ETRelayData.cpp", find subroutine `ProcMsgDataV0,` and find the comment `// Get "verg_gaze_coord_x"`

Delete the lines under this comment (directly after the `"FlagDsp = "` statement) that put a value in the `Sample` buffer. In other words delete the following lines:

```
if (FlagDsp)
      pSample[SIndex++] = (float)ValDouble;
else
      pSample[SIndex++] = 0;
```

Repeat this under the `// Get "verg_gaze_coord_y"` and `// Get "verg_gaze_coord_z"` comment lines.

To add an item to the data set, increment the `NUM_ARGUS_LSL_CHANNELS,` on the ETRelayData.h file, to the appropriate number. Add the desired item to the list at the beginning of ETRelayData.cpp. Find the comment in the `ProcMsgDataV0` subroutine to "Get" the desired data item. Below the "`FlagDsp = `" statement, add the code to put a value in the `Sample` buffer.

Be sure items in the "LSL Streaming Channels list always appear in the same order as on the *ETVision, Data Selection* dialog list (and the "// Get " items in the `ProcMsgDataV0` subroutine). When adding a value to the Sample buffer be sure to use the correct data type ("(float)ValDouble", or "(float)ValUInt").