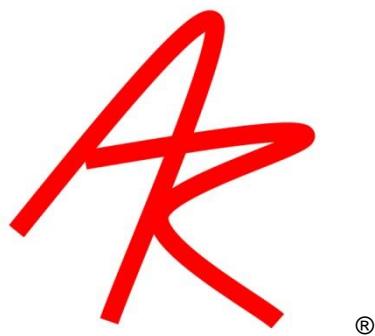


ViewPoint EyeTracker®

UserGuide



Arrington Research

Karl Frederick Arrington, Ph.D.



ViewPoint-UserGuide-117(b).docx

June 28, 2019 14:35:00

2019 © Arrington Research, Inc.
All rights reserved.

Contact Information

Arrington Research, Inc.
27237 N 71st Place, Scottsdale, AZ 85266
United States of America

Phone +1-480-985-5810

www.ArringtonResearch.com

ViewPoint-EyeTracker@ArringtonResearch.com

ViewPoint EyeTracker ® is a Registered Trademark of Arrington Research, Inc.
ViewPoint ~ Voltage ™ is a Trademark of Arrington Research, Inc.
RemoteLink™ is a Trademark of Arrington Research, Inc.
ViewPoint Client ™ is a Trademark of Arrington Research, Inc.
3D ViewPoint ™ and **3DVP**™ are Trademarks of Arrington Research, Inc.
3D WorkSpace ™ and **3DWS**™ are Trademarks of Arrington Research, Inc.
Experiment Engine ™ is a Trademark of Arrington Research, Inc.

ViewPoint EyeTracker version: **2.9.5.146**

X:\ARI-Shared\ARI Graphic Arts\ARI Documentation (Shared)\ViewPoint UserGuide (Shared)\ViewPoint-UserGuide-117(b).docx

CHAPTER 1. INTRODUCTION

1

1.1 Congratulations	1
1.2 Contact Us	1
1.2.1 Custom Software & Hardware Development	1
1.2.2 User Feedback & Support	1
1.3 License Information and Conditions of Use	1
1.4 Styles	2
1.5 High-Risk Activities Warning	3
1.6 Computer System Requirements	3
1.7 How to Use this <i>UserGuide</i>	3
1.8 Hardware Installation	4
1.9 Documentation Folder	4
1.10 Info Window	4
1.11 Citing <i>ViewPoint</i>	5
1.11.1 Citing <i>ViewPoint</i> in Methods Section	5
1.11.2 Citing <i>ViewPoint</i> in References Section	5

CHAPTER 2. OVERVIEW OF VIEWPOINT

6

2.1 Complete Environment	6
2.2 Interfaces	6
2.2.1 Graphical User Interface (GUI)	6
2.2.2 Command Line Interface (CLI)	6
2.2.3 Using CLI Commands	7
2.2.3.1 Settings Files	7
2.2.3.2 Interactive CLI Window	7
2.2.3.3 FKey and TTL	7
2.2.3.4 SDK/API	7
2.2.3.5 StateEngine	7
2.2.4 Software Developer's Kit (SDK)	7
2.2.5 Ethernet & <i>ViewPointClient</i> ™	8
2.2.5.1 <i>ViewPointClient</i> ™ for PC	8
2.2.5.2 <i>ViewPointClient</i> ™ for Mac	8
2.3 Interfaces to Third Party Products	8
2.3.1 Integrity Suite™	8
2.3.2 Other Third Party Interfaces	8
2.4 Use the SDK to write your own Layered Applications.	9
2.4.1 SDK Communication	9
2.4.2 Example Layered Interface Applications	9
2.5 Pilot Data	9

CHAPTER 3. DIFFERENT CALIBRATION METHODS

10

CHAPTER 4. SOFTWARE INSTALLATION
11

4.1 ViewPointLicense (.VPL) File	12
4.2 Program Layout	12
4.2.1 User Windows	12
4.2.2 Menu Navigation	15
4.3 Criteria	16

CHAPTER 5. QUICK START SECTION
17

5.1 Eye Camera & LED Illuminator Positioning	17
5.1.1 Camera Positioning	17
5.1.2 EyeFrame Camera Positioning	18
5.1.3 Remote System Camera Positioning	18
5.2 Pan/Zoom/AutoCenter	18
5.2.1 Panning	18
5.2.2 Zooming	19
5.2.3 AutoCenter	19
5.3 Locating the Pupil – All Systems	20
5.3.1 Corrective Lenses (Eye Glasses)	20
5.4 Thresholding – All Systems	21
5.5 Calibration	21
5.5.1 Calibration (Head Fixed and HMD Systems)	21
5.5.1.1 HMD Partial Binocular Overlap	22
5.5.2 Calibration (<i>SceneCamera</i>)	22
5.6 Stimuli	24
5.6.1 Choosing the Type of Stimulus	24
5.6.2 <i>ViewPoint Stimulus</i> Window	24
5.6.3 Interactive Computer Display	24
5.6.4 <i>SceneCamera</i> Systems	24
5.6.5 Show SceneVideo in <i>Stimulus</i> window	25
5.7 Data Collection – All Systems	25
5.8 Recording Scene and Screen Movies	27
5.8.1 Recorded Movie Format	27
5.8.2 Display of Overlay Graphics in Recorded Movie	28
5.8.3 Compression	28
5.9 DataAnalysis program	28
5.10 Analysis Options	28
5.10.1 Head Fixed and HMD Systems	28
5.10.2 Playing Scene and Screen Movies	29
5.11 Frequently Used Settings	29
5.12 Preferred Window Layout	29
5.13 Accelerator Keys & FKeys	29
5.14 Printing	29



6.1 EyeCamera Window	31
6.2 Feature Method	31
6.2.1 Single DataPoint	31
6.2.2 Multiple DataPoint	31
6.2.2.1 Advantages	32
6.2.2.2 Disadvantages	32
6.2.3 Slip Compensation	32
6.3 Simulation of Gaze	32
6.3.1 Manual Simulation	32
6.3.2 Pattern Simulation	33
6.4 Thresholding	33
6.5 Setting the Scan Density	33
6.6 Manual Thresholding of the Dark Pupil	34
6.7 Step-by-step guide for <i>Glint-Pupil Vector</i> method	35
6.8 Noise	36
6.9 Automatic Slip Compensation	36
6.10 Feature Criteria	37
6.10.1 Pupil Aspect Criterion	37
6.10.2 Width Criteria	37
6.11 Alternative Segmentation Methods	37
6.11.1 Ellipse	37
6.11.2 Centroid	38
6.11.3 Oval Fit	38
6.11.4 PupilScanArea Shape Options	38
6.11.5 Glint Segmentation Methods	38
6.11.6 Edge Trace (only on special versions)	38

7.1 To switch Operating in Binocular Mode	39
7.2 CLI Prefix – EyeTarget Specifier	39
7.3 Setup	40
7.4 Storing Data	40
7.5 Real-Time Display of Binocular Data	40

8.1 Calibration Carryover	41
8.2 Choosing the Number of Calibration Points	41
8.3 Automatic Calibration (Head Fixed)	41
8.4 Assessing Calibration Success	42



Arrington Research

8.5 CalibrationImage	43
8.6 Binocular Calibration	44
8.7 Omitting Individual Calibration Points	44
8.7.1 Automatic Omitting	44
8.7.2 Manual Omitting	45
8.8 Re-presenting Individual Calibration DataPoints	45
8.9 Slip Correction	45
8.10 Gaze Nudge	45
8.11 Dominant Eye	46
8.12 Advanced Calibration Controls	46
8.12.1 Calibration Stimulus & Background Color	47
8.12.2 Timing & Warning	47
8.12.3 Presentation Order	47
8.12.4 Stimulus-Point Locations	48
8.12.4.1 Custom Calibration Point Positions	49
8.12.4.2 Partial Binocular Overlap	49
8.12.5 Flipping the Initial Calibration	49
8.12.6 Adjusting the Calibration Area	50
8.12.7 Snap and Increment Calibration Modes	50
8.12.8 Manual Calibration	50
8.12.9 Calibration Points in a <i>Settings</i> file	52

CHAPTER 9. CORRECTIONS BASED ON MEASUREMENTS 53

9.1 Obtaining POG in Degrees: 2D Geometry	53
9.1.1 Stimulus Window or InterActive Display -- HeadFixed (non-HMD)	54
9.1.2 HeadMounted SceneCamera	54
9.1.3 HeadMounted Display	55
9.1.4 Notes on Measurement Lines	56
9.1.5 Geometry Grid	56
9.2 Parallax Correction for Binocular SceneCamera systems	57
9.2.1 Manual Parallax Adjustment	58
9.2.2 Parallax Correction from Data	58
9.3 Pupil Diameter	59
9.3.1 Raw Pupil Size	59
9.3.2 Calculated Pupil Diameter	60
9.4 Inter-Pupillary Distance (IPD)	61
9.5 Pupil Aspect	61

CHAPTER 10. CURSOR CONTROL - EYEMOUSE 62

CHAPTER 11. OCULAR TORSION 63



Arrington Research

11.1 Introduction to Torsion	63
11.2 Procedure for Measuring Torsion	65
11.3 Torsion Demonstration Test	66
11.4 Cyclovergence	66
11.5 Overriding the Default Torsion Parameters	66
11.6 Torsion Data	67
CHAPTER 12. STIMULUS PRESENTATION (HEADFIXED)	68
12.1 Experimental Control	68
12.2 Image Display	68
12.3 PictureList	69
12.4 Using the Stimulus Window (Head Fixed Option)	70
12.5 Stereo Display	71
12.6 Regions of Interest (ROI)	73
12.7 ROI use Corrected Data	74
12.8 Associating an Image with Specific ROI	75
12.9 ROI Transition Statistics or Linkages	76
CHAPTER 13. DATA VISUALIZATION & ANALYSIS	78
13.1 Real-Time	78
13.1.1 <i>GazeSpace & GraphicsOptions</i>	79
13.1.2 PenPlots	80
13.1.3 Data Smoothing	80
13.2 Fixation, Saccade, Drift and Blinks	81
13.2.1 Fixations	81
13.2.2 Criteria Levels	82
13.2.3 Saccade Velocity Criterion	82
13.2.4 Drift Criterion	83
13.2.5 Fixation Time Criterion	83
13.2.6 Blinks	83
13.2.7 Events Markers	83
13.2.8 Events Window	83
13.2.9 SDK	84
13.3 Post-Hoc	84
13.4 Summary Data	84
CHAPTER 14. DATA COLLECTION	85
14.1 Sampling Rate	85
14.2 Saving Data to File	85
14.2.1 Unprocessed & Corrected Data	85



Arrington Research

14.2.2 AlsoMake	86
14.3 Data File Format	86
14.3.1 File Header Information	86
14.3.2 File Records	86
14.3.3 Synchronous vs. Asynchronous Data Inserts	87
14.3.4 Data Record Tags.	88
14.3.5 UserData	89
14.3.6 User Defined Tags	90
14.4 PupilWidth and PupilHeight calculations	94
14.5 Direction-of-Gaze Coordinates	94
14.6 Raw Data	94
14.7 Timing Measurement	94
14.8 Display Screen Geometry	95
14.9 Events Data	95
14.10 Regions of Interest (ROI)	96
14.11 Quality Marker Codes	96

CHAPTER 15. STATE ENGINE 97

15.1 State Engine Commands	97
15.1.1 State Initialization	97
15.1.2 State Setup	97
15.1.3 State Commands	97
15.1.4 State Transition Commands	98
15.1.5 State Space Debug	98
15.2 Picture Lists	98
15.3 Counters	99
15.4 Miscellaneous	100

CHAPTER 16. USING SETTINGS FILES 102

16.1 CLI String Parsing	102
16.2 Saving and Loading <i>Settings</i> Files	102
16.2.1 <i>Startup</i> Folder	103
16.2.2 <i>FinishUp</i> Folder	103
16.2.3 <i>Settings/LastRun.txt</i>	103
16.3 <i>Settings</i> File Examples	103
16.4 CLI commands	104
16.5 Associating CLI s with FKeys	104
16.6 Command Line Interface Window	104
16.7 <i>Settings</i> File Lists (Deprecated)	104

CHAPTER 17. ETHERNET COMMUNICATION BETWEEN COMPUTERS 106



Arrington Research

17.1 Ethernet Software Connections	107
17.1.1 Changing the <i>ViewPoint</i> IP Address	107
17.1.2 Changing the Port Number	107
17.1.3 Running the <i>ViewPoint</i> Server	108
17.1.4 Ping Clients	108
17.1.5 Loopback	108
17.1.6 Static IP Addresses and Zero Configuration	108
17.1.7 Firewalls	109
17.1.8 <i>ViewPointClient</i> & <i>ViewPoint</i> on the same machine	109
17.1.9 <i>ViewPointClient_32 interface for 32-bit applications</i>	109
17.2 Ethernet Hardware Connections	110
17.2.1 Hub, Switch, Router, or a Crossover Cable?	110
17.2.2 Direct Connection using Ethernet Cable.	110
17.3 Ethernet to <i>Microsoft Windows</i> computers	110
17.3.1 How to use <i>ViewPointClient</i>	110
17.3.2 <i>Windows</i> Third Party Applications	112
17.3.3 Layered Applications	112
17.4 Ethernet to <i>Apple Macintosh</i> computers.	113
17.4.1 <i>Macintosh</i> Third Party Applications	115
17.4.2 Terminal Window (deprecated method?)	115
17.5 Ethernet Server Error	116
17.6 ViewPoint ClientTest - CommandLineTool	117

CHAPTER 18. COMMAND LINE INTERFACE (CLI) 118

18.1 Example of a Command Line:	118
18.2 Important CLI Changes from Previous Versions	119
18.3 Arguments	119
18.4 Asynchronous Operations	120
18.5 Error Detection and Reporting	120
18.6 Parameters and Arguments	120
18.7 Using Commands	121
18.7.1 Interactive CLI Window	121
18.7.2 FKey and TTL	121
18.7.3 Settings Files	121
18.7.4 SDK/API	121
18.8 Boolean Toggle	121
18.9 Quoting Strings	122
18.10 White Spaces	123
18.11 Case Insensitive CLI Strings	123
18.12 Embedded Special Characters	123
18.13 SDK Return Values	123
18.14 VPX_SendCommand & Formatted Strings	123
18.15 TargetPrefix / EyePrefix	124



19.1 General	125
19.2 Help finding CLI commands	125
19.2.1 Help	125
19.3 Data Files	126
19.3.1 Specify NewUnique Data File Extension	126
19.3.2 NewUnique DataFile Name	127
19.3.3 Open a Data File and Specify a File Name	128
19.3.4 DataFile AlsoOpen	128
19.3.5 Insert a String into the Data File	129
19.3.6 Insert a Marker into the Data File	130
19.3.7 Insert a User Defined Tag into the Data File	131
19.3.8 Specifies Asynchronous or Synchronous String Data	131
19.3.9 Specify Asynchronous or Synchronous Marker Data	132
19.3.10 Specify Asynchronous or Synchronous Head Tracker Data	132
19.3.11 Specify Data File Start Time	133
19.3.12 Include Raw Eye Data in Data File	133
19.3.13 Include Events Data File	133
19.3.14 Specify Whether to Use <i>DataFile</i> Buffering (DEPRECATED)	134
19.3.15 Pause Writing of Data to File	134
19.3.16 Opening Data File in Paused State	135
19.3.17 Close Data File	135
19.3.18 Close Data File and Open in Post-Hoc Analysis tool	135
19.3.19 DataAnalysis Application	136
19.4 Corrected Data	136
19.4.1 Geometry Window	136
19.4.2 Geometry Measurement	136
19.4.3 Geometry Grid Spacing	137
19.4.4 GeometryGrid Lines Display	137
19.4.5 Specify Amount of Parallax Correction	137
19.4.6 Inter-Pupillary Distance (IPD) Measure	138
19.4.7 Pupil Diameter Calibration	138
19.5 Stimulus Images	139
19.5.1 Load Stimulus Image into the Stimulus window	139
19.5.2 Specifies How to Display the Currently Loaded Stimulus Image	139
19.5.3 Specify a Background "Matting" Color for the Stimulus Window	140
19.5.4 Stereo Display Mode (Side-by-Side)	140
19.5.5 Create a Fake, or Pseudo, Stereo Image (Side-by-Side)	140
19.5.6 Swap EyeA and EyeB for Stereo Image	141
19.5.7 Play specified Sound File	141
19.6 PictureList	142
19.6.1 Initialize Picture List	142
19.6.2 Add New Image Name to Picture List	142
19.6.3 Randomize List of Images in the Picture List	142



19.6.4	Move to Next Image in the PictureList	142
19.6.5	Move to Start of Images in Picture List	143
19.6.6	Picture List End Action	143
19.7	Controls Window: EyeImage	143
19.7.1	Specify Mapping Feature	144
19.7.2	AutoThreshold	144
19.7.3	Positive Lock Tracking	144
19.7.4	Adjust Pupil Threshold Slider	145
19.7.5	Adjust Glint Threshold Slider	146
19.7.6	Adjust Video Image Brightness	146
19.7.7	Adjust Video Image Contrast	147
19.7.8	Camera Shutter / ExposureTime	147
19.7.9	Dynamically Optimize Brightness and Contrast Settings	148
19.7.10	Adjust Pupil Scan Density	149
19.7.11	Override Pupil Scan Density Minimum	150
19.7.12	Adjust Glint Scan Density	151
19.7.13	Override Glint Scan Density Minimum	152
19.8	EyeCamera Window	153
19.8.1	Adjust Pupil Scan Area	153
19.8.2	Pupil AutoCenter	153
19.8.3	Specify Pupil Scan Area Shape	154
19.8.4	Pupil and Glint Oval Fit Constraints	154
19.8.5	Define Glint Scan Area	155
19.8.6	Define Offset of Glint Scan Area Relative to the Pupil	155
19.8.7	Unyoke Glint Scan Area from the Pupil	156
19.8.8	Define Offset of Unyoked Glint Scan Area	156
19.8.9	<i>EyeCamera</i> and <i>EyeMovie</i> Overlay Graphics Options	157
19.8.10	EyeCamera Tool Bar Display	158
19.9	Slip Compensation	159
19.9.1	Slip Compensation Speed	159
19.9.2	Slip Compensation X-Gain & Y-Gain	159
19.10	EyeMovie related controls	160
19.10.1	Open / Close new EyeMovie file.	160
19.10.2	EyeMovie Play	160
19.10.3	EyeMovie Load ...	161
19.10.4	EyeMovie EndAction	161
19.10.5	EyeMovie Percent Location	161
19.10.6	EyeMovie Play Speed	162
19.10.7	EyeMovie Binocular	162
19.10.8	EyeMovie Binocular	162
19.10.9	EyeMovie Zoom / Pan	163
19.11	Video related controls	164
19.11.1	Specify EyeCamera Video Input Standard	164
19.11.2	Specify SceneCamera Video Input Standard	165
19.11.3	Specify Video Operation Mode	165



19.11.4	Specify Dark or Bright Pupil Tracking	166
19.11.5	Specify Pupil Segmentation Method	166
19.11.6	Specify Glint Segmentation Method	167
19.11.7	Toggle Freeze Video Image Preview On / Off	167
19.11.8	Reset Video Capture Device	168
19.11.9	VideoMirror	168
19.12	Calibration Controls	169
19.12.1	Start Auto-Calibration	169
19.12.2	Specify Calibration Stimulus Presentation Speed	169
19.12.3	Specify the Duration of Calibration Warning Notice	170
19.12.4	Specifies Interval Between Presentation of Calibration Points	170
19.12.5	Specify Number of Calibration StimulusPoints	171
19.12.6	Specify Calibration StimulusPoint Color	171
19.12.7	Specify Calibration Stimulus Window Background Color	171
19.12.8	Specify CalibrationStimulus Type	172
19.12.9	Calibration Snap Mode	172
19.12.10	Calibration Beep when Finished	173
19.12.11	RePresent in Snap Calibration Mode	173
19.12.12	AutoIncrement Calibration Mode	174
19.12.13	Calibration StimulusPoint Presentation Order	175
19.12.14	Specify Calibration StimulusPoint Presentation Order	175
19.12.15	Specify Individual Custom Calibration StimulusPoints	176
19.12.16	Display Custom Calibration StimulusPoint Order	176
19.12.17	Select the Specified Calibration DataPoint	176
19.12.18	Specify an Index Number for Calibration DataPoint	177
19.12.19	Calibration StimulusPoint Location Method	178
19.12.20	Specify Custom Calibration StimulusPoint Locations	179
19.12.21	Dump Custom Calibration StimulusPoints Locations	180
19.12.22	Display Nearest-Neighbor Gridlines in <i>EyeSpace</i> Window	180
19.12.23	Compensate for Slip	180
19.12.24	Adjust Calibration Area	181
19.12.25	Undo the Last Operation on a Calibration DataPoint	181
19.12.26	Re-Present the Specified Calibration DataPoint	181
19.12.27	Save Image of Eye at Each Calibration DataPoint	182
19.12.28	Calibration Quality	182
19.12.29	Nudge	183
19.12.30	Manual Calibration	183
19.12.31	Initialial Calibration Flip (Mirroring)	184
19.13	Controls: Criteria Controls	185
19.13.1	Specify amount of Smoothing	185
19.13.2	Specify Smoothing Algorithm to Apply	185
19.13.3	Specify Velocity Threshold	186
19.13.4	Specify amount of Drift Allowed	186
19.13.5	Specify Fixation Time Criterion	187
19.13.6	Specify Pupil Minimum AspectRatio Failure Criterion	187



19.13.7	Specify Pupil Maximum AspectRatio for PupilAngle to Change.	188
19.13.8	Specify Pupil Min & Max Size Failure Criterion	188
19.14	Region of Interest (ROI)	189
19.14.1	Define an ROI Box	189
19.14.2	Set ROI Name	189
19.14.3	Draw IsoEccentric	189
19.14.4	Remove all ROI Boxes	190
19.14.5	Select a Specific ROI	190
19.14.6	Select the Next ROI Box	190
19.14.7	Lock ROI Settings	191
19.14.8	Associate ROI with a Particular Stimulus Image	191
19.14.9	Set ROI Shape	191
19.15	PenPlot Controls	192
19.15.1	<i>ViewPoint</i> PenPlot names and their meanings	192
19.15.2	<i>3DViewPoint</i> & <i>3DWorkSpace</i> PenPlots	192
19.15.3	PenPlot Dump Names	193
19.15.4	Specify Which PenPlot Traces to Display	193
19.15.5	PenPlot Background Color	194
19.15.6	PenPlot Limen Fill Color	194
19.15.7	Specify Speed of PenPlot Scrolling	194
19.15.8	Specify Size of PenPlot Lines	195
19.15.9	Specify Range of PenPlot Values	195
19.15.10	Restart the PenPlot	195
19.15.11	Specify the Behavior of the PenPlot after resuming from a VideoFreeze	196
19.16	Graphics Controls	196
19.16.1	Specify the Eye Color for GUI graphics	196
19.16.2	GazePoint Transparency	197
19.16.3	Graphics Options for <i>GazeSpace</i> & <i>Stimulus</i> windows and <i>SceneMovie</i>	197
19.16.4	Erase Data Displays in the <i>GazeSpace</i> and Stimulus Windows	198
19.16.5	Automatically Erase Display Windows	198
19.16.6	Specify Time Delay for Auto Erase	198
19.17	Stimulus Window Controls	199
19.17.1	Specify Stimulus Source	199
19.17.2	Specify Custom Stimulus Window Size and Position	200
19.17.3	Automatically Show the Stimulus Window upon Calibrate	201
19.17.4	Show SceneVideo in <i>Stimulus</i> window	201
19.17.5	Specify How and Where to Show Stimulus Window	202
19.18	Window Related Controls	203
19.18.1	Size Window	203
19.18.2	Move Window, or Move & Resize Window	203
19.18.3	Specify <i>ViewPoint</i> Window State, or Windows Layout	204
19.18.4	<i>History</i> Window Options	205
19.18.5	<i>History</i> User Text	205
19.18.6	Clear <i>History</i> Window	205
19.18.7	Clear Events Window	206



Arrington Research

19.18.8	GazeSpace MouseAction	206
19.19	Settings File Commands	206
19.19.1	Load Settings File	206
19.19.2	Edit Settings File	207
19.19.3	Verbose CLI Parsing and loading of Settings Files	207
19.19.4	Save Settings e.g. Calibrations etc.	207
19.19.5	Save Window Layout Settings	208
19.20	SettingsFileList (DEPRECATED. Please use the <i>StateEngine</i>)	208
19.20.1	Initialize Settings File List	208
19.20.2	Next Settings File in List	209
19.20.3	Add Settings File to the List	209
19.20.4	Restart Settings File List	209
19.20.5	Toggle Autosequencer ON / OFF	209
19.20.6	Specify Delay between Settings Files in List	210
19.20.7	Randomize Settings Files	210
19.21	Torsion Commands	210
19.21.1	Start / Stop Torsion Calculations	210
19.21.2	Adjust Radius of Torsion SamplingArc	211
19.21.3	Adjust Start Point of Torsion SamplingArc	211
19.21.4	Adjust Length of Torsion SamplingArc	211
19.21.5	Autoset Torsion Template after Adjustments	212
19.21.6	Display Real-Time Torsion Data	212
19.21.7	Adjust Torsion Measurement Range	213
19.21.8	Adjust Torsion Measurement Resolution	213
19.21.9	Set Autocorrelation Template	213
19.22	Interface Settings Commands	213
19.22.1	GazeCursor On / Off	214
19.22.2	GazeCursor Transparency	214
19.22.3	Turn Cursor Control On / Off	214
19.22.4	Use Fixation to Issue Button Click	215
19.22.5	Specify Fixation Time to Issue Button Click	215
19.22.6	Use Blinks to Issue Button Click	215
19.23	Ethernet	216
19.23.1	Ethernet Server	216
19.23.2	Ethernet Port Number	216
19.23.3	Ethernet IP Address	216
19.23.4	Ethernet List IP Addresses	217
19.23.5	Ethernet Ping Clients	217
19.24	Binocular Commands	217
19.24.1	Turn Binocular Mode On / Off	217
19.24.2	Specifies Binocular Averaging	218
19.24.3	Specifies which Eye to Calibrate	218
19.25	System Files & Applications Related	219
19.25.1	Launch Application with Command Line Options	219
19.25.2	SystemOpen	220



Arrington Research

19.25.3	Quit <i>ViewPoint</i>	220
19.25.4	Confirm Quit	220
19.25.5	Specify Default Folder Paths	221
19.26	FKey	222
19.26.1	Associate CLI's with FKeys	222
19.27	TTL	223
19.27.1	Associate CLI's with TTL Voltage Changes	223
19.27.2	Set TTL Output Voltages	223
19.27.3	Simulate Change in TTL Input	224
19.27.4	Print TTL Values in the <i>History</i> Window	224
19.27.5	Set TTL Output to Indicate Data Quality Codes	225
19.28	Misc.	226
19.28.1	Specify Verbose Information to Send to <i>History</i> Window	226
19.28.2	Status Dump	226
19.28.3	Update Eye Data on Request	227
19.28.4	Set Status Window Update Rate for FPS Field	227
19.28.5	SDK Debug Mode	227
19.28.6	Priority	228
19.28.7	Specify <i>ViewPoint</i> Generated Events	228
19.29	Parser Instructions	228
19.29.1	Comment	228
19.29.2	End of Settings File Command	229

CHAPTER 20. SOFTWARE DEVELOPERS KIT (SDK) & API **230**

20.1	General	230
20.2	RealTime Callback Functions	230
20.3	Registering to Receive WindowMessages (Deprecated)	231
20.3.1	Example SDK Code (Deprecated method)	232
20.4	Data Quality Codes	232
20.5	Sending CLI's via the SDK	232
20.6	High Precision Timing	233
20.7	DLL Version Checking	233
20.8	SDK Trouble Shooting	234
20.8.1	Different DLLs	234
20.8.2	Different DLL Versions	234
20.9	SDK Data Access Functions	234

CHAPTER 21. SDK FUNCTIONS **235**

21.1	GazeData	235
21.1.1	GetGazePoint	235
21.1.2	GetGazeBinocular	236
21.1.3	GetGazeAngle	237



Arrington Research

21.1.4	GetTotalVelocity	238
21.1.5	GetComponentVelocity	238
21.1.6	GetVelocityBinocular	239
21.2	3D-Data	240
21.2.1	GetHeadPositionAngle	240
21.2.2	GetPanelHit	241
21.2.3	GetVergenceAngle	241
21.2.4	GetGazePoint3D	242
21.2.5	GetVersionAngle	242
21.2.6	GetVersionComponentVelocity	243
21.2.7	GetVersionTotalVelocity	243
21.3	Eye Events	244
21.3.1	GetFixationSeconds	244
21.3.2	GetDrift	244
21.3.3	GetBlinkEvent	245
21.3.4	GetEyeMovementEvent	245
21.4	ROI	246
21.4.1	GetROI_RealRect	246
21.4.2	ROI_GetHitListLength	246
21.4.3	ROI_GetHitListItem	247
21.4.4	ROI_MakeHitListString	248
21.5	EyeSpace	249
21.5.1	GetPupilSize	249
21.5.2	GetPupilAspect	250
21.5.3	GetPupilOvalRect	250
21.5.4	GetPupilAngle	251
21.5.5	GetPupilDiameter	251
21.5.6	GetPupilPoint	252
21.5.7	GetPupilCentroid	252
21.5.8	GetDiffVector	253
21.5.9	GetGlintPoint	253
21.5.10	GetGlintCentroid	254
21.5.11	GetTorsion	254
21.5.12	Data Quality	255
21.6	Time Stamps	256
21.6.1	GetDateTime	256
21.6.2	GetDataDeltaTime	256
21.6.3	GetStoreTime	257
21.6.4	GetStoreDeltaTime	257
21.6.5	Precision Timing	258
21.7	Miscellaneous	259
21.7.1	GetCursorPosition	259
21.7.2	GetStatus	260
21.7.3	GetViewPointHomeFolder	260
21.7.4	GetMeasuredViewingDistance	261



21.7.5	GetMeasuredScreenSize	261
21.8	Calibration Information	262
21.8.1	VPX_CalibrationEventRecord	262
21.8.2	GetCalibrationEventRecord	262
21.8.3	GetCalibrationStimulusPoint	263
21.9	HWND Functions	264
21.9.1	Set Remote EyeImage	264
21.9.2	Set EyeImage Display Rectangle	264
21.9.3	Set External Stimulus Window	265
21.9.4	GazeSpace / Stimulus Window	266
21.10	CallbackFunction Interface	267
21.10.1	Insert Callback function	267
21.10.2	Remove Callback function	268
21.10.3	List Callback functions	268
21.10.4	Get Layered App CallbackListLength	268
21.10.5	VPX_CallbackResult	269
21.11	MessageRequest Interface (Deprecated)	270
21.11.1	Insert MessageRequest	270
21.11.2	Remove MessageRequest	270
21.11.3	Remove non-responding MessageRequests	271
21.11.4	Get Message List Length	271
21.11.5	Get Message Post Count	271
21.11.6	GetViewPointAppCount	272
21.12	Version Checking and Matching for SDK / DLL	272
21.12.1	VPX_GetDLLVersion	273
21.12.2	VPX_VersionMismatch	273
21.12.3	VPX_GetRevisionNumber	273
21.13	SDK Utility Functions	274
21.13.1	DebugSDK	274
21.13.2	Draw Rectangle	274
21.13.3	Draw Ellipse	274
21.13.4	Draw ROI	274
21.13.5	Convert: pixelRect → normalizedRect	275
21.13.6	Convert: normalizedRect → pixelRect	275
21.13.7	Convert: LParam → RealPoint	275
21.13.8	Convert: LParam → RectPoint	276
21.14	Events and Notification Messages	277
21.14.1	General Events	277
21.14.2	Calibration Events	280
21.15	Structures and Enumerations	285
21.15.1	VPX_PositionAngle	285
21.15.2	VPX_GetImageRecord	285
21.15.3	VPX_QUALITY_	285
21.15.4	VPX_GLINT_QUALITY	285
21.15.5	VPX_STATUS_	286



Arrington Research

21.15.6	VPX_BinocularAveragingType	286
21.15.7	VPX_DistributorType	286
21.15.8	VP_Message_NotificationCodes	287
21.15.9	VPX_ParseType	288
21.15.10	VPX_CallbackResult	288
21.15.11	VPX_EyeDataRecord	289
CHAPTER 22. TROUBLESHOOTING		290
22.1	<i>History</i> Window	290
22.2	Improving Frame Rate	290
22.3	EyeCameraWindow Troubleshooting	290
22.3.1	Bottom Half of EyeCamera Window is Black (Analog 60 Hz products)	290
22.4	General Troubleshooting	291
CHAPTER 23. ERROR CODES		292
23.1	Introduction to Error Codes	292
CHAPTER 24. HARDWARE INSTALLATION		296
24.1	USB-220 & USB-400 Installation	296
24.2	60 Hz Video Capture	298
24.2.1	Camera Systems	298
24.2.2	Using with Third Party Video Cameras & Signals	298
24.3	USB-60x3 (SilverBox) Installation and Set Up	299
24.4	PC-60 (PCI card) Installation and Set Up	300
CHAPTER 25. LATENCY		303
CHAPTER 26. SAFETY		304
26.1	Infrared Light	304
CHAPTER 27. ARI SOFTWARE LICENSE		305
CHAPTER 28. THIRD PARTY LICENSES		307
CHAPTER 29. APPENDIX		308
29.1	Diagram of the Eye	308



Arrington Research

309

309

29.2 Mapping to GazeSpace

29.3 Schematic Bloc Description of *ViewPoint*

Chapter 1. Introduction

1.1 Congratulations

Congratulations on your purchase of the *ViewPoint EyeTracker* ®. It has been designed to be the easiest to use, most reliable, and best value eye tracker on the market. Almost 20 years of development and improvement have made the *ViewPoint EyeTracker* ® the most powerful eye tracking environment available.

- ✓ Intuitive User Interface
- ✓ Powerful SDK allows easy integration with other application programs
- ✓ Powerful Command Line Interface allows extensive customization & easy configuration
- ✓ Free interfaces to popular 3rd party programs, e.g., MATLAB, Presentation, E-Prime, etc.
- ✓ Built-in State Machine for experimental control
- ✓ Pupilometry
- ✓ AnalogOut and TTL options
- ✓ Torsion option

It provides:

- ✓ A comprehensive solution for eye tracking research.
- ✓ An embeddable eye tracking solution for third party products and end user custom applications.

1.2 Contact Us

Feel free to contact us for support, ideas, custom projects, business opportunities, etc.
Please email inquiries to: ViewPoint-EyeTracker@ArringtonResearch.com

1.2.1 Custom Software & Hardware Development

Special software and hardware development for laboratories or organizations may be performed under individual consulting agreements. *ViewPoint EyeTracker* ® is easily customized as an embedded eye tracking solution for OEMs. We also help OEMs to interface the *ViewPoint EyeTracker* ® with their equipment by providing custom camera and optical solutions.

1.2.2 User Feedback & Support

Suggestions for improvements to this *ViewPoint UserGuide* or to the *ViewPoint EyeTracker* ® software are always welcome and appreciated. Please report any problems as soon as possible, a work-around may be easy for you, but may not be so clear for another user.

1.3 License Information and Conditions of Use

Use of the software constitutes consent to the terms of the “ARI Software License” in described in [Chapter 27](#). The contents of this *UserGuide* and any other documentation provided with the *ViewPoint EyeTracker* ® are for the use of registered *ViewPoint EyeTracker* ® users only. No part of this or other *ViewPoint EyeTracker* ® documentation or Software Developer Kit (SDK) information may be distributed or shared with others, without prior written permission from Arrington Research, Inc.

1.4 Styles

The meaning of different type fonts is specified in [Table 1](#), and the meaning of different shading and icons are specified in [Table 2](#).

Table 1. Meaning of Type Fonts	
<i>Type Font</i>	<i>Example Meaning</i>
Eye tracking has many applications.	Body
<code>dataFile_NewUnique</code>	CLI command (Calibri, 10, Bold, OrangeRed)
<code>VPX_GetGazePoint2 (EYE_B, &gazePoint);</code>	Code (Courier, 9, MidnightBlue)
<i>File > Data > New Data File</i>	GUI controls: Menu, etc. (Calibri, 10, Italic, IntenseBlue)
<i>Controls</i> window	Window name (Italic)
Table 19	Link within document or Hyperlink (Underline, TealBlue)
RemoteLink™	Deprecated or obsolete (gray color)

Table 2. Meaning of Shading and Icons	
<i>Example</i>	<i>Meaning</i>
 Read the <i>UserGuide</i> .	Hint box
 Do not start your experiment without first collecting and understanding test data!	Warning Box
 The default setting is 0.1	Note box

1.5 High-Risk Activities Warning

Every effort has been made to provide a bug-free product. Nevertheless, this software is not intended for use in the operation of nuclear facilities, aircraft navigation, communications systems, air traffic control, medical treatment and diagnosis, or for any other use where the failure of the software could lead to death, personal injury, or damage to property or the environment.

1.6 Computer System Requirements

The *ViewPoint EyeTracker* ® runs on the Windows-7, Windows-8, and Windows-10 operating systems. Please make sure that the latest service packs are loaded.

1.7 How to Use this *UserGuide*

New users should study Chapters 2, 3, and 4 to get started:

[Chapter 2: Overview of ViewPoint](#) describes the principle-of-operation of the *ViewPoint EyeTracker*® and provides an overview of its design.

[Chapter 5: Quick Start Section](#) provides a brief tutorial designed to help new users start recording eye movements very quickly and easily.

[Chapter 18: Command Line Interface \(CLI\)](#) is the most comprehensive reference section for all aspects of the eye tracker and should be referred to often. It contains every feature and control available, including those with no corresponding GUI. The remaining chapters each deal with a different task or set of tasks that the user may want to perform, and they provide some helpful background on eye tracking.



1.8 Hardware Installation

The *ViewPoint EyeTracker* ® uses a variety of different video capture hardware. Hardware and driver installation is described in [Chapter 24](#). In particular find the following sections:

Table 3. Installation Sections for Specific Products	
<i>Section</i>	<i>Products</i>
24.1	Digital Camera Products: USB-90, USB-220, USB-400
24.2	General information on 60 Hz Analog Camera products
24.3	USB-60x3 = USB SilverBox
24.4	PC-60 = PCI card (soon to be discontinued)

1.9 Documentation Folder

The *Documentation* Folder contains many files besides this *ViewPoint EyeTracker* ® *UserGuide*. The second most important file is the *ChangeHistory* that usually includes many new features that have not yet made it into this *UserGuide*, as well as detailing important changes.



Use menu item: [Help > Documentation ...](#) to quickly access the Documentation folder.

In addition, documentation for various layered products is included inside the specific product folder inside the `~/ ViewPoint/Interfaces/` folder. Refer below for examples:

```

ViewPoint/Interfaces/Voltage - Digital InOut & Analog Out/
ViewPoint/Interfaces/3rdParty/ViewPoint EyeTracker Toolbox .pdf
ViewPoint/Interfaces/3rdParty/Microsoft-Windows/MATLAB/
ViewPoint/Interfaces/3rdParty/Microsoft-Windows/Python/
ViewPoint/Interfaces/3rdParty/Microsoft-Windows/E-Prime/
ViewPoint/Interfaces/3rdParty/Microsoft-Windows/LabView/
ViewPoint/Interfaces/3rdParty/Microsoft-Windows/NBS-Presentation/
ViewPoint/Interfaces/3rdParty/Apple-Mac-OSX/ViewPointMacX-nnn-nnn.zip
ViewPoint/Interfaces/3rdParty/Apple-Mac-OSX/MATLAB/
ViewPoint/Interfaces/3rdParty/Apple-Mac-OSX/ViewPoint-ClientTest/

```

1.10 Info Window

The *Info* window can be displayed using menu item [Help > Info](#). This provides system information, a list of keyboard shortcut keys, Function Key (FKey) assignments, display devices that *ViewPoint* has detected, the computer IP address, default folder paths, etc.

1.11 Citing *ViewPoint*

We ask that you give credit to the *ViewPoint EyeTracker* ® in both the Methods Section and the References Section of any published papers.



Note that ***ViewPoint*** is one word with only the letters V and P capitalized, and that ***EyeTracker*** is also one word, with only the letters E and T capitalized. It is a registered trademark and should be followed by a capital R within a circle, ®, or if not available, a capital R within parenthesis, (R).

1.11.1 Citing *ViewPoint* in Methods Section

Please use the following format when citing the *ViewPoint EyeTracker* ® in the Methods Section of your papers.

***ViewPoint EyeTracker* ® by Arrington Research, Inc. (www.ArringtonResearch.com)**

Please also include the web site address (www.ArringtonResearch.com), where ArringtonResearch is one word. Correct capitalization is not required for the web link to work properly, however using only a capital A and a capital R is the preferred form and it makes the link more readable.

We would love to hear about your research, published or not, please let us know what you are working on!

1.11.2 Citing *ViewPoint* in References Section

Please also give credit in the References Section as:

**Arrington, Karl Frederick (2016) "ViewPoint EyeTracker® UserGuide"
Arrington Research, Inc., Scottsdale, Arizona U.S.A.**

Chapter 2. Overview of *ViewPoint*

2.1 Complete Environment

The *ViewPoint EyeTracker* ® provides a complete eye movement evaluation environment, including Fixation, Saccade, and Drift classification, Region of Interest events, Integrated Stimulus Presentation, Simultaneous Eye Movement and Pupil Diameter Monitoring, and a *Software Developer's Kit* (SDK) for communicating with other applications. It incorporates several methods from which the user can select to optimize the system for a particular application. Basic *DataAnalysis* capabilities are also included.

2.2 Interfaces

There are many ways to interface to the *ViewPoint EyeTracker* ® for data synchronization, communication and control:

- ④ Graphical User Interface (GUI)
- ④ Command Line Interface (CLI)
- ④ Software Developers Kit (SDK)
- ④ Ethernet & *ViewPointClient* ™
- ④ Third Party applications

2.2.1 Graphical User Interface (GUI)

The most common controls are exposed as buttons, sliders, menu items etc. Please be aware that these are only a subset of the complete set of features and controls available through the CLI.

2.2.2 Command Line Interface (CLI)

The *ViewPoint EyeTracker* ® provides a *Command Line Interface* (CLI) that allows users to control almost every aspect of the program.

There are many more CLI commands than there are GUI controls in *ViewPoint EyeTracker* ®. For example there are commands to allow fine control of *ViewPoint EyeTracker* ® operations and behavior. There is a command for each GUI control. For example, to adjust the amount of data smoothing, to display / hide various pen plots, freeze / un-freeze the eye camera video. There are commands to open, pause, resume, and close *ViewPoint EyeTracker* ® data files. There are commands to insert remote synchronization data into the *ViewPoint EyeTracker* ® data files. If there is an action that you need to perform but cannot find a GUI control for it, refer to [Chapter 19](#) to see if a CLI command exists. The syntax for using CLI commands is described in [Chapter 18](#).



2.2.3 Using CLI Commands

There are many ways to send CLI commands to *ViewPoint*, these include:

2.2.3.1 Settings Files

Every *Graphical User Interface* (GUI) selection and adjustment that the user makes in *ViewPoint* (e.g., menu item selection, radio button selection, slider value) can be saved in a *Settings* file as a *Command Line Interface* (CLI) text string instruction, called a CLI command. These *Settings* files can then be loaded at a later time. The control values are stored as single line ASCII commands in the form of a keyword and parameters. When a *Settings* file is loaded, each line in the file is sent to the *Command Line Parser* (CLP).

2.2.3.2 Interactive CLI Window

Select the Menu Item: *Windows > Command Line Interface* to raise a window in which you may type and issue CLI commands directly to *ViewPoint*. Just type in the command and press the *Send* button. [Chapter 18](#) describes the CLI commands in detail.

2.2.3.3 FKey and TTL

Most keyboards have a set of *Function Keys*, called *FKeys* for short, e.g.: *F1*, *F2*, etc. For convenience, CLI commands can be associated with *FKeys*, and also with TTL inputs (with the TTL option). The associations are shown in menu: *Help > Info > Shortcuts tab, or TTL Cmd tab*.

2.2.3.4 SDK/API

CLI command strings can also be sent to *ViewPoint* from other programs while *ViewPoint* is running, which means that outside, “layered”, programs can have complete control of the *ViewPoint EyeTracker*®. These command strings can be sent via the *Software Developers Kit* (SDK) with the function `VPX_SendCommand("some command string")`. This can be done from programs running on the same machine or from programs running on remote computers via Ethernet and *ViewPointClient*.

2.2.3.5 StateEngine

ViewPoint contains a *StateEngine* in that can be setup to issue CLI commands whenever a state is entered or exited. See [Chapter 15](#) for details.

2.2.4 Software Developer's Kit (SDK)

The *ViewPoint* software package includes a powerful *Software Developer's Kit* (SDK) that allows seamless, real-time interfacing with the *ViewPoint EyeTracker*. The SDK (a) provides real-time access to all *ViewPoint* data, (b) provides for calibration stimuli in the user's *Stimulus* window, or for the user's application to draw into the *ViewPoint Stimulus* and *GazeSpace* windows, (c) allows complete external control of the *ViewPoint EyeTracker*®. The SDK interface is based on shared memory in a *Dynamic Link Library* (DLL). The SDK is event / message driven, so there is no CPU load from polling and provides microsecond latency.

The SDK includes a consistent *Application Programming Interface* (API) for MS Windows and MAC. All CLI commands can be sent via the `VPX_SendCommand ("CLIcommand")` function. Data is obtained via access functions of the form `VPX_GetSomething (&dataVariable)`; see [Chapter 20](#).



2.2.5 Ethernet & *ViewPointClient*™

2.2.5.1 *ViewPointClient*™ for PC

ViewPointClient™ software runs on a remote computer and communicates with *ViewPoint*. The client software uses a dynamically linked library to exchange data just like *ViewPoint* does, but typically takes less than one percent of CPU resources. This means that the same “layered” applications can be used on a remote computer just as easily as on the same computer. *ViewPoint* contains an Ethernet server; the *ViewPointClient* software establishes an Ethernet link with this server. *ViewPointClient* for PC is a separate program, and is included free with *ViewPoint*. See [Chapter 17](#) for details.

2.2.5.2 *ViewPointClient*™ for Mac

The *ViewPointClient* for Mac is no longer a separate program; it has been integrated into the DLL-based SDK, which simplifies the interaction. The layered application on the Mac calls `VPX_ConnectToViewPoint` to establish the client/server communication; see section [17.4](#) for details.

2.3 Interfaces to Third Party Products

2.3.1 Integrity Suite™

Integrity™ is a suite of interfaces between the *ViewPoint EyeTracker*® and third party applications. This suite is provided by *Arrington Research, Inc.*, to ensure professional quality, uniform, complete, and thorough interfaces to the favorite products of our users. These interfaces provide access to data, complete eye tracker control, and data integration and synchronization, all in real-time. *Integrity* is included free with the *ViewPoint* product.

Interfaces include:

- ④ **MATLAB** ® = *ViewPoint EyeTracker*® toolbox (PC & Mac)
- ④ **E-Prime** ® = EBasic interface
- ④ **LabView** ®
- ④ **Python** ® (PC & Mac)
- ④ **Presentation** ® = Binocular plugin for *Neurobehavioral Systems*
- ④ **Gazetracker** ®

2.3.2 Other Third Party Interfaces

Many third party applications provide the means to load our DLL into their program, which allows a user to call *ViewPoint* SDK functions from within these applications. These applications include:

- ④ **Vizard** ® by WorldViz
- ④ **SimCreator** by Realtime Technologies
- ④ **PsyScope** ®
- ④ **SuperLab** ® by Cedrus
- ④ **g.Tec** ® products

See our website for an ever increasing list of third party applications and hardware that work with *ViewPoint*.

Finally, some vendors also interface to the *ViewPoint EyeTracker* using the ***ViewPoint~Voltage***™ option that provides Analog Voltage and TTL outputs. For example:

2.4 Use the SDK to write your own Layered Applications.

2.4.1 SDK Communication

Layered applications communicate seamlessly with the *ViewPoint EyeTracker* either on the same machine, or on remote machines via Ethernet, using the *ViewPoint* SDK.

Separate documentation and examples are provided in the folder [~/ViewPoint/SDK/](#). *ViewPoint* users are encouraged to write and share their own layered applications with the user community. We will be more than happy to assist.

2.4.2 Example Layered Interface Applications

The folder [~/ViewPoint/Interfaces/VPx64-Client/](#) and the folder [~/ViewPoint/Interfaces/VPx32-Client/](#) contain already compiled executable application programs from the example projects from the [~/ViewPoint/Interfaces/Programming/SampleCode/](#) folder. Try running the example layered applications to demonstrate communication with *ViewPoint* either directly via the DLL on the same machine, or indirectly via the DLL and [ViewPointClient_nn.exe](#) software running on the same, or a different, computer. On a *Windows PC*, simply, copy the sample interface applications from the main [~/ViewPoint/](#) folder; on a *Mac* use the sample application in the *Mac* [~/ViewPoint/Interfaces/3rdParty/Apple-Mac-OSX/](#) folder.

For the *Windows PC*, these may include the sample applications listed below, which will Open, Pause, Resume, and Close data files; display streaming x and y position data, and also include a means to send CLI commands.

- [VPX_MFC_Demo.exe](#) –
- [VPX_Win32_Demo.exe](#) –
- [VPX_Basic_Demo.exe](#) –
- [DataMarker.exe](#) – This program allows easy manual insertion of data file marker characters into an open *ViewPoint* data file. (*Not included in all releases*).

The functionality is the same as that via menu item: [Windows > KeyPad / DataMarker](#).

Note that it is impossible to launch the *ViewPoint EyeTracker* application using these interface applications on remote machines, because the client must already be connected to *ViewPoint* (rather than to the local DLL). The [Stop ViewPoint button](#) will work.

2.5 Pilot Data

It is extremely important that you read and understand this *UserGuide*. Chapters 4, 5, 6, 9, 10 and 11 will get you started. Do not wait until your subjects are waiting. Calibrate and collect data on yourself first. Always collect and analyze pilot data before running your experiments so that you understand exactly what you are getting.

Chapter 3. Different Calibration Methods



Using an incorrect calibration procedure will give you unusable data.

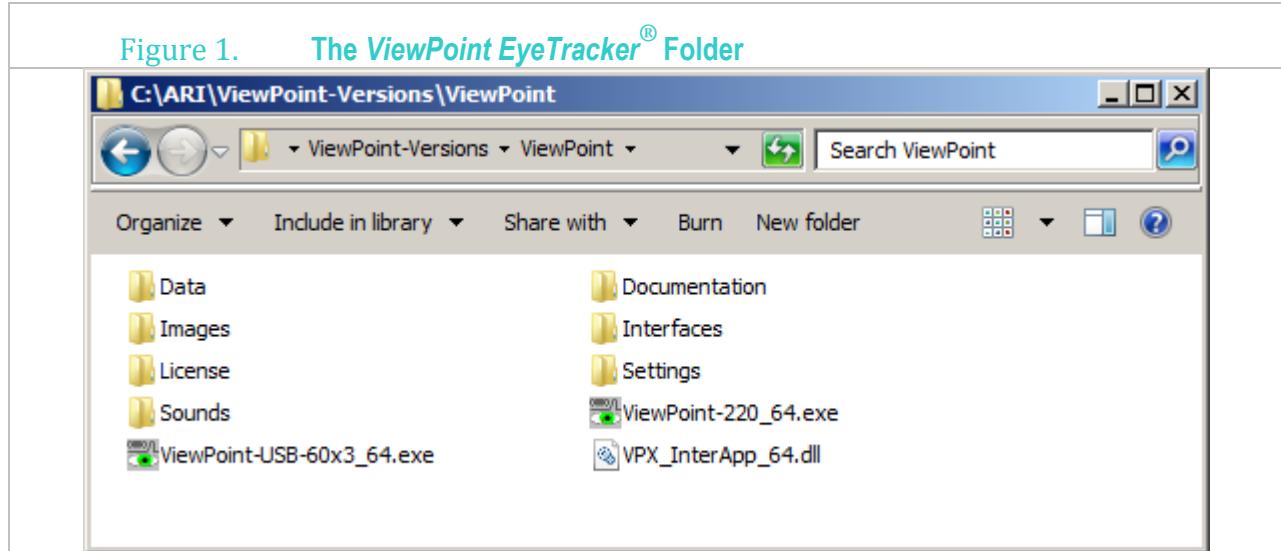
System	<i>Head-Free SceneCamera™</i>	<i>Head-Fixed HeadLock™ Remote Systems</i>	<i>Head-Mounted Display (HMD)</i>
Description	The <i>SceneCamera</i> systems allow gaze position superimposed on real-world scene video. No head tracker is required.	<i>Head-Fixed</i> systems are designed for visual psychophysics, etc. that require a stable viewing distance and position, or restriction of head movement.	<i>HMD</i> systems are perfect for Virtual Reality, Augmented Reality, etc.
Calibration	To <i>SceneCamera</i> viewing area. Refer to section 5.5.2 The calibration is with respect to the <i>SceneCamera</i> sensor that moves with the subject, NOT the scene image content that can change.	Automatic On-Screen Refer to section 5.5.1 Calibration is performed with respect to a computer screen or projector display.	Automatic On-Screen Refer to section 5.5.1 Calibration is performed with respect to the HMD display.
Output	ASCII format files showing eye movement relative to the <i>SceneCamera</i> viewing area. Movie files of the <i>SceneCamera</i> video showing the eye movement superimposed on it.	ASCII format files showing Position of Gaze relative to the display screen used during calibration. Can make movie of screen (e.g. web page) and eye movement, interaction.	ASCII format files showing eye movement relative to the HMD screen.

Chapter 4. Software Installation

When you purchased the *ViewPoint EyeTracker* you were provided with a link to download the latest version of the software from the ArringtonResearch web site. Clicking the link should automatically start the download.

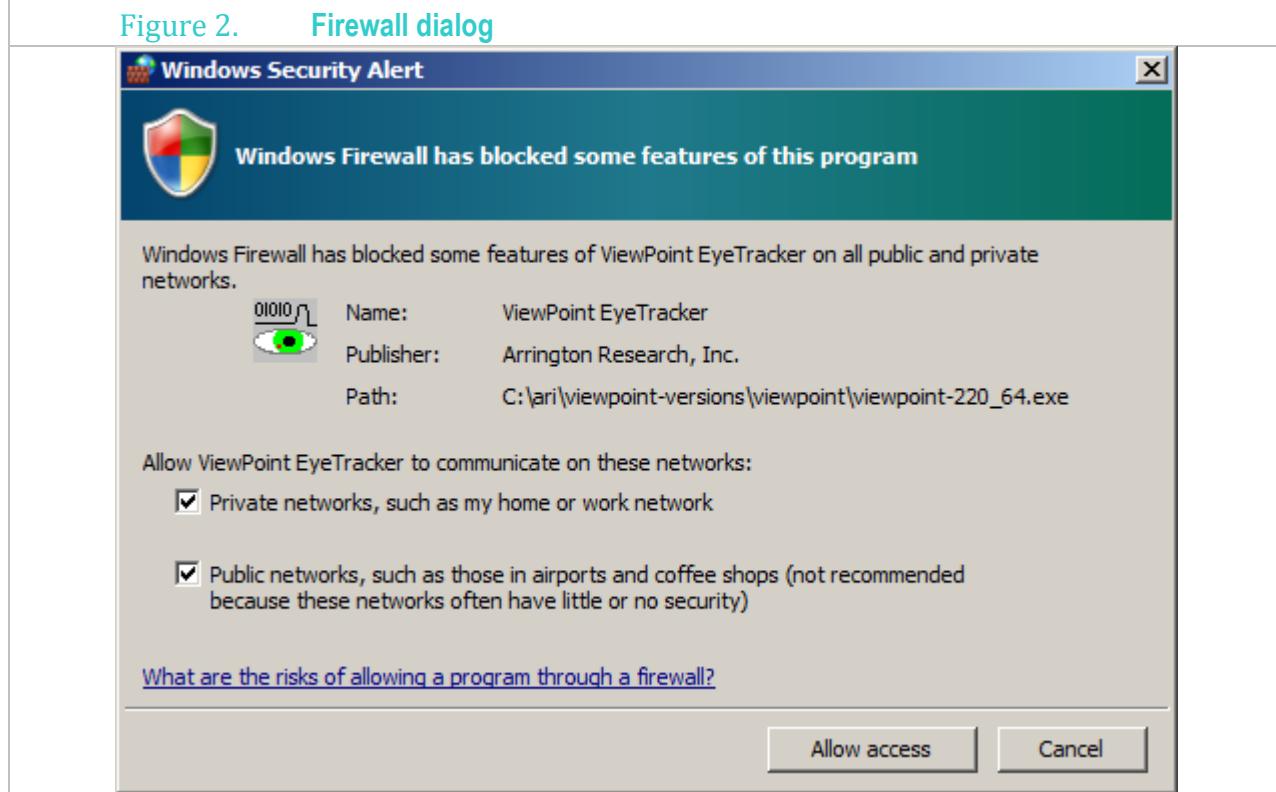
Older distribution mailed a CD-ROM disk. If you have a disk please copy the *ViewPoint* folder from the CD-ROM to the hard drive of your computer. The program will not work properly if you try to run it from the CD-ROM.

The basic *ViewPoint* folder is illustrated in [Figure 1](#). Various products may provide additional required DLL files. The directory structure supplied must be maintained for proper functioning of the software. *ViewPoint* will not run without the [VPX_InterApp_64.dll](#) file.



You may start the program immediately by double clicking the *ViewPoint* application program. The first time the program is launched from a new location the Windows OS will provide a Windows Security Alert regarding the Windows Firewall. Please make sure to allow *ViewPoint* to communicate on all networks in order to avoid Ethernet communication problems with the server built into the *ViewPoint EyeTracker*; see [Figure 2](#).

Figure 2. Firewall dialog



4.1 ViewPointLicense (.VPL) File

The video capture boards and USB cameras (and optional hardware) contain serial numbers that must match one of the numbers encrypted into your *ViewPointLicense (.vpl)* file. This *.vpl* file is located in the ...\\ViewPoint\\License\\ folder. The license file also specifies any options that may have been enabled. Your license file is named in the format *YourName.vpl* or *12345678.vpl*. *ViewPoint* will also look in the *C:\\Windows\\ViewPoint\\License* folder.

4.2 Program Layout

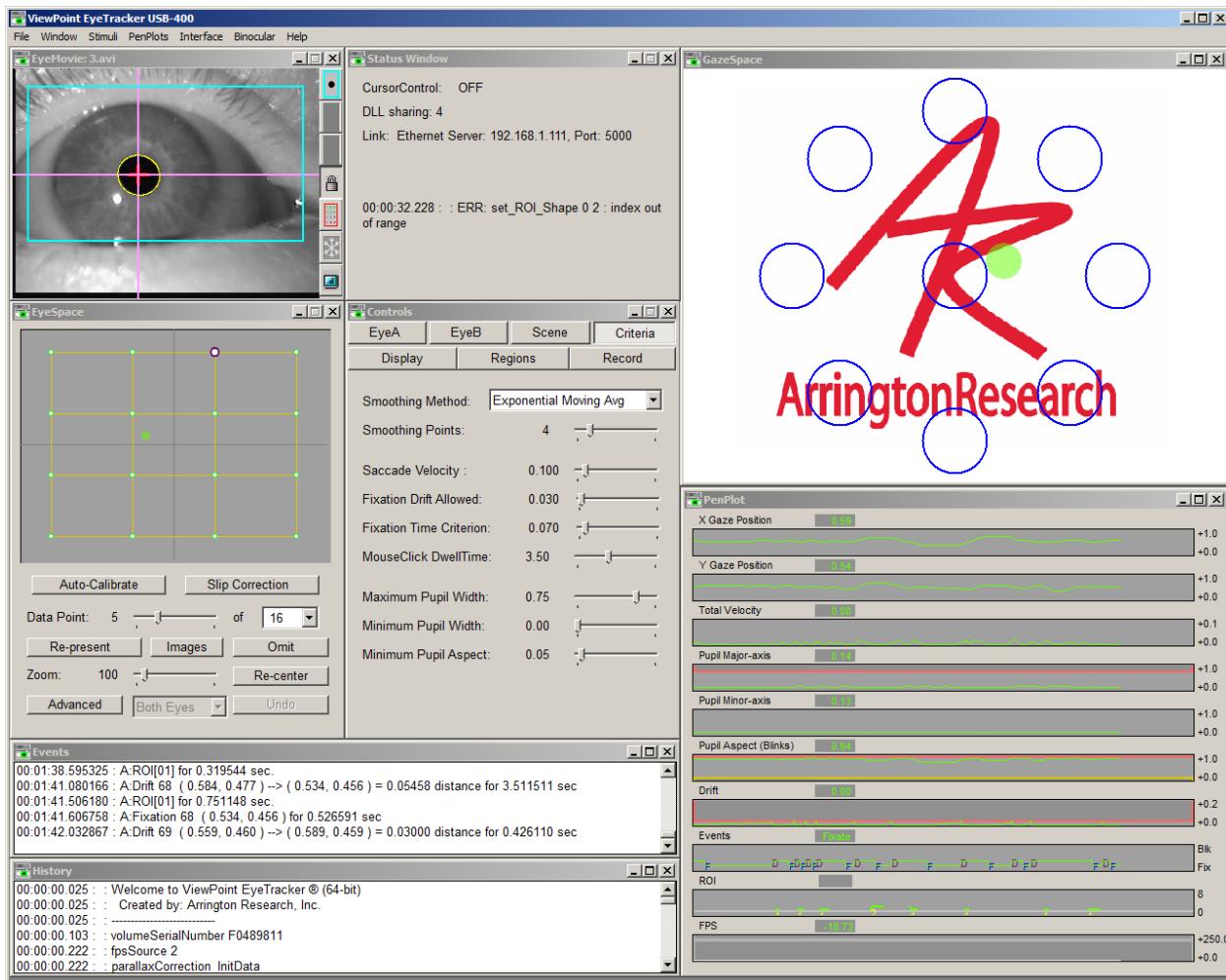
The window layout may be customized. The starting window layout depends upon the detected display size.

Most *ViewPoint* windows are normally attached to the *ViewPoint* parent window (they are child windows), but they may be set free, which allows, for example, the *PenPlot* window to be full screen on a secondary display.

4.2.1 User Windows

When *ViewPoint* is started, it displays several windows arranged as shown below.

Figure 3. Start-up Arrangement of the User Windows



Note: The startup window layout depends upon the detected display size. Users can specify their own layout or load program defied layouts, e.g. CLI: **layout 1280x1024**, or those defined in *Settings* files: ...\\ViewPoint\\Settings\\Layouts\\ or select menu item: *Windows > Arrange > Larger Layout SXGA (1280x1024)*.

To use multiple displays, you may need to install a second display card into your computer and consult the computer's operating manual for configuration settings.

The window **ControlMenu** is the usual pull down menu in the upper left corner of every window. *ViewPoint* adds functionality to some of its window ControlMenus, such as: **Free/Child toggle**, and **Clear Window**. This is particularly useful in the **PenPlot**, **History**, and **GazeSpace** windows.

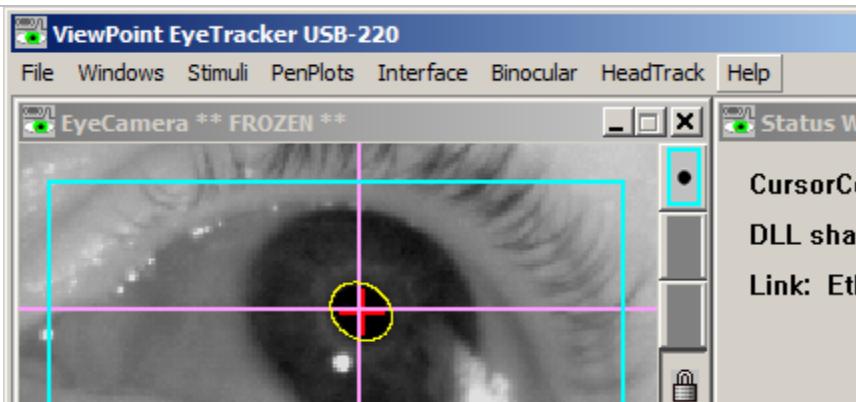
Table 4. Window Descriptions

Window	Description
<i>EyeCamera</i>	Displays the video image of the eye and image analysis graphics.
<i>EyeSpace</i>	Displays calibration controls and calibration results. The relative locations of the pupil, glint, or difference vector, which were obtained during calibration, are shown in the graphics at the top of the window. The graphics rectangles (one for monocular, two for binocular) corresponds to the geometry of the <i>EyeCamera</i> video image. They provide information about calibration accuracy and allow rapid identification and correction of individual calibration errors by allowing manual recalibration of individual points or the ability to omit problem points. Refer to Chapter 8 .
<i>GazeSpace</i>	Displays a miniature representation of the <i>Stimulus</i> window, allowing the experimenter to monitor eye movement in real-time.
<i>Controls</i>	Provides easy-access control of most common operations, grouped under the tabs: <i>EyeA</i> and <i>EyeB</i> tabs: Eye Image quality adjustments and tracking method specification. <i>Criteria</i> tab: Specify smoothing and other criteria to apply to the data. <i>Display</i> tab: Specify information to be displayed in the <i>Stimulus</i> and <i>GazeSpace</i> windows. Also to specify information to be displayed in recorded SceneMovies for users with the <i>SceneCamera</i> option. <i>Regions</i> tab: Specifies the <i>GazeSpace MouseAction</i> , including specification of: Regions of Interest (ROIs), the Calibration Region, Gaze Nudge, Calibrate on Content, Simulation, etc. <i>Scene</i> tab: (<i>SceneCamera</i> option only) Adjust brightness, contrast, hue, and saturation etc. of scene image; checkbox to show the SceneVideo in <i>Stimulus</i> window. <i>Record</i> tab: Quick and easy way to open, pause, close, and analyze data files as well as insert markers.
<i>Events</i>	Displays completed events and their durations, e.g., Saccades, Fixations, etc.
<i>History</i>	Provides feedback about CLI parsing, system performance, errors, etc.
<i>Stimulus</i>	Displays the stimulus image that the subject views, which is designed to be presented full screen, preferably on a second display. This window can also display the subject's eye movement, ROI, etc., as on the <i>GazeSpace</i> window. Refer to Chapter 12 .
<i>Penplot</i>	Displays real-time line graphs and numerical data values of, for example: X and Y gaze, velocity, torsion, pupil width, ROI hits, etc. The user may select which PenPlots to display using menu item <i>PenPlot > *</i> . The range of many of the PenPlot displays can be adjusted by clicking the right mouse button in the various penPlot graphics wells.

4.2.2 Menu Navigation

The *ViewPoint* main menu bar is at the top of the *ViewPoint* application window, under the Title bar.

Figure 4. The ViewPoint Main Menu Bar



Menu Item	Use this Menu to:
<i>File ></i>	Open, pause and close data files, Save and load <i>Settings</i> files, Load Images
<i>Windows ></i>	Open and close windows, specify window layout. <i>Shift-click</i> to bring a window to the front.
<i>Stimuli ></i>	Specify the type of stimuli to be used, Control the <i>Stimulus</i> window settings, and Specify the <i>GeometryGrid</i> settings
<i>PenPlots ></i>	Hide or show various penPlots, and Specify PenPlot window settings
<i>Interface ></i>	For GazeCursor and Eye Moves Mouse features, and For the Ethernet server interface.
<i>Binocular ></i>	Binocular EyeMovement Settings. Dimmed if Binocular Option not purchased.
<i>Help ></i>	Access system configuration information, View license Holder information, options, etc. Access the <i>Documentation</i> folder, and Link to the www.ARRINGTONRESEARCH.com web site

4.3 Criteria

ViewPoint provides several criteria for accepting or rejecting data. Using these wisely and judiciously can substantially improve the performance in many situations. These criteria can help tag noisy blink data so that it can be easily eliminated from post-hoc analysis. They can also improve real-time performance and help protect equipment; for example when a galvanometer is controlled by the eye positions signals.

These criteria can be divided into two categories: *Feature Criteria* and *Movement Criteria*.

Table 5. Criteria	
<i>Feature Criteria</i>	<i>Movement Criteria</i>
<i>Minimum Pupil Width</i>	<i>Fixation Drift Allowed</i>
<i>Maximum Pupil Width</i>	<i>Saccade Velocity</i>
<i>Minimum Pupil Aspect</i>	<i>Fixation Time Criterion</i>

There are also criteria for certain actions, such as the *dwellTime* criterion for issuing a system mouse-click event.

The new user should be able to precede through the **Quick Start** sections in this *UserGuide* without needing to adjust these criteria. However, it is important to understand them prior to undertaking data collection. Further information is provided in later sections.

Chapter 5. Quick Start Section

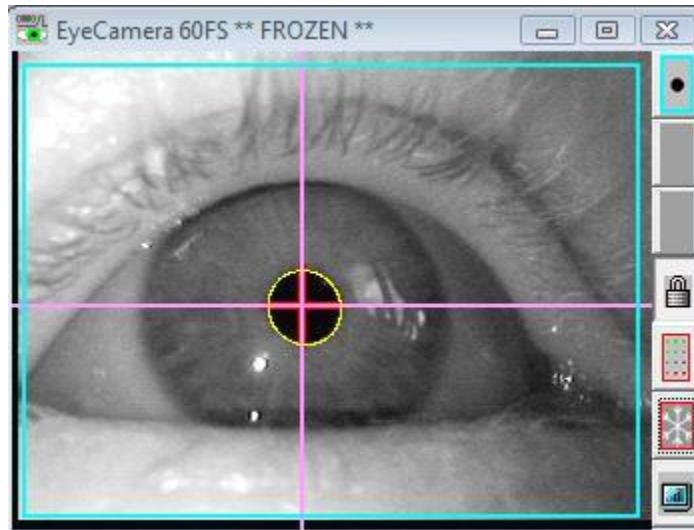
This chapter contains all of the information necessary for a new user to quickly and easily get the system up and running. For simplicity, it assumes that only the *Pupil Location* method is used. However, for normal subject testing, the *Glint-Pupil Vector* method may be more appropriate (refer to [Chapter 6.](#))

5.1 Eye Camera & LED Illuminator Positioning

This section describes the proper positioning of the Camera & LED. Refer to section [5.1.1](#) for Head Fixed systems and to section [5.1.2](#) for *EyeFrame* systems. In all cases the goal is to obtain an image of the eye similar to that shown below in [Figure 5](#). In general, the eye should just fit, such that the corners of eye should be at the sides of the window.

In general, an EyeCamera should not be more than 45 degrees below the “straight-ahead” line-of-sight.

Figure 5. EyeCamera Window



Generally, the image of the eye should be such that the corners of the eye (the canthi) are at the horizontal edges of the *EyeCamera* window. The camera may be closer in if the gaze is not eccentric.

5.1.1 Camera Positioning

Instructions for proper positioning of the Camera & LED when using the *HeadLock™* or *QuickClamp™* hardware:

1. Position the camera about 40 to 45 degrees below the subject’s line-of-sight (as viewed from the side), and at the appropriate distance from the eye (which for the USB 220 cameras mounted on the *HeadLock* is approximately 15 cm.)



Arrington Research

2. Position the LED so that it appears at the 11 o'clock position for the right eye and 1 o'clock for the left eye, when looking at the camera lens, such that the top surface of the LED and the camera lens are along the same horizontal plane. This allows both the LED and the camera to simultaneously be as high as possible, while not occluding the vision of the monitor.
3. Move the camera mount sideways, such that the LED is centered along the optical axis of the eye while the eye is looking in the center of the display. This will result in the camera being slightly off axis (as viewed from above). In other words, if the subject looks straight down she should be looking at the LED, not at the camera lens. Relative movement between the head and the *EyeCamera* must be minimized, which is the purpose of the *Arrington Research Precision Head Positioners* (*QuickClamp™* and *HeadLock™*) and camera system.

5.1.2 EyeFrame Camera Positioning

1. Place the *EyeFrame* glasses on the subject.
2. Tighten the strap to minimize movement on the head.
3. Use the collar clip to reduce cable weight and twisting (clip may be attached to the tail of the head strap).
4. Adjust the position of the *EyeCamera* to obtain an image of the eye such that the pupil is in the center of the *EyeCamera* window when the subject is looking straight ahead, and the eyeball fills the maximum possible area, as shown in [Figure 5](#).
5. The lens on an *EyeFrame* *EyeCamera* can often be rotated manually by pinching the part of the lens extending from the housing. (Older models use the two holes on the lens front surface.)

5.1.3 Remote System Camera Positioning

As remote systems are sold to accommodate varying operating setups, exact positioning of the camera and illuminator will also vary, depending, for example, on the lens chosen. Position both the camera and illuminator to achieve a well illuminated image of the eye that fills the whole *EyeCamera* window.

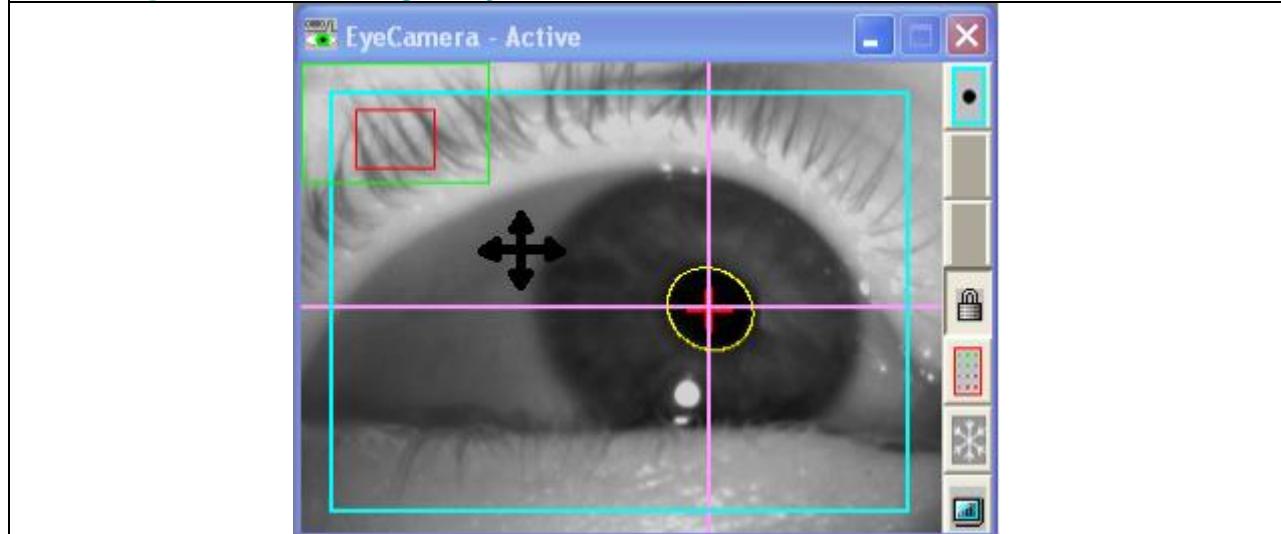
5.2 Pan/Zoom/AutoCenter

Some systems (USB-220, including some other frame rates) allow software Pan, Zoom and pupil AutoCenter.

5.2.1 Panning

Holding the Right mouse button down in the *EyeCamera* window changes the cursor icon to a “move” or “drag” icon indicating that the Area Of Interest (AOI) can be panned left/right or up/down. The relative size and location of the AOI is shown in the upper left corner of the *EyeCamera* window; the AOI is shown as a red rectangle inside a green rectangle that represents the camera sensor; see [Figure 6](#). During Setup, we recommend positioning the AOI in the center, before moving the actual camera to locate the subject’s eye.

Figure 6. Panning the EyeCamera AOI



5.2.2 Zooming

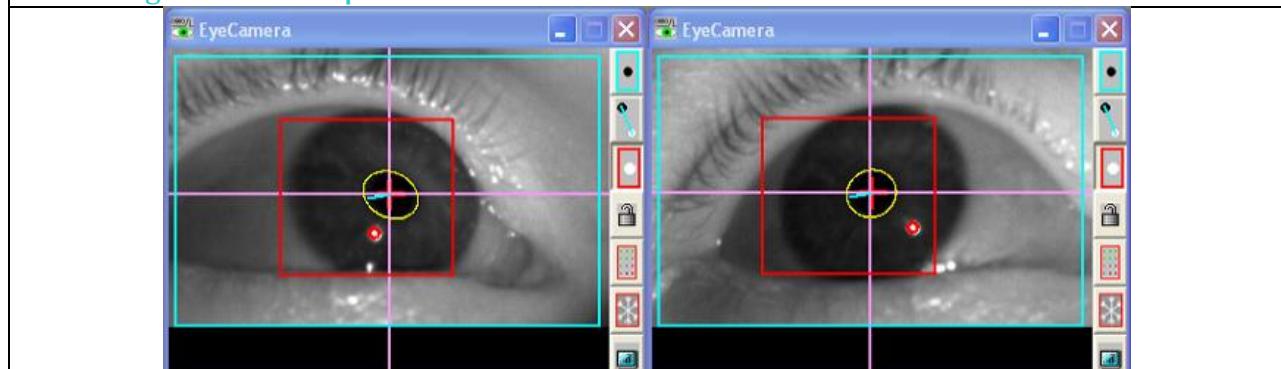
The mouse wheel controls the zoom.

5.2.3 AutoCenter

AutoCenter tries to keep the pupil of the eye in the center of the *Pupil Scan Area* that is shown as the cyan rectangle in the EyeCamera window. *AutoCenter* automatically pans the AOI in the EyeCamera sensor. [Figure 7](#) shows two images of a subject's right eye at different gaze positions with pupil *AutoCenter* enabled. In both cases, *ViewPoint* has centered the pupil within the *Pupil Scan Area*.

Pupil *AutoCenter* is only useful, and thus only available, for the Glint-Pupil Vector feature method. The subject should calibrate with the *Glint-Pupil Vector* feature method selected. Pupil *AutoCenter* is not required during the calibration, but can be enabled during the calibration if desired. Once the calibration is complete, the user can enable and disable pupil *AutoCenter* by using CLI command: **pupilAutoCenter ON**

Figure 7. Pupil AutoCenter



5.3 Locating the Pupil – All Systems

The following steps are applicable to all systems:

- a. Adjust the camera so that the pupil is centered in the *EyeCamera* window as the subject looks at the center of the display and the eyeball fills the maximum possible area as shown in [Figure 5](#).
- b. Defocus the camera so that the corneal glint is spread to about the size of an eighth (or more) that of the pupil. Besides making the glint larger, defocusing also effectively lowers the intensity of small bright extra reflections (by virtue of the point spread function). Defocusing may be achieved by rotating the camera lens or by adjusting the camera closer or farther from the eye. Generally, the image of the eye should be such that the corners of the eye (the canthi) are at the horizontal edges of the camera window.
- c. Many third party LEDs will produce a doughnut shape of illumination. If that is the case, adjust the LED so that the darker center of the doughnut is in the center of the camera image, this will put the brighter ring around the edges near where the canthi and lids are located. This doughnut may be more easily observed by placing the palm of the hand, or a piece of paper, at the location of the eye and then moving the LED slightly.



ViewPoint systems are designed to produce a uniform diffuse IR illumination and you should not see the doughnut effect.

- d. If the video image is too dark or too bright you can adjust the contrast and brightness settings by adjusting the [Controls window > Brightness and Contrast sliders](#). When adjusting the brightness and contrast controls in *ViewPoint*, the general goal is to increase the range of gray levels as far as possible, that is, to DECREASE the contrast as much as possible, while MAXIMIZING the blackness of the pupil and the whiteness of the glint. However, the glint should be the only spot that is saturated to maximum brightness.

The [EyeCamera window > Monitor icon > Histogram window](#) can help with optimizing brightness and contrast.

- e. Decrease the brightness until you obtain a pupil that is as black as possible and adjust the contrast so that the glint, and only the glint, is of maximum white.
- f. [Autolimage](#) may help, especially in varying light conditions. See CLI command: `videoAutolimage`.

5.3.1 Corrective Lenses (Eye Glasses)

There are two main effects relating to the fact that both the front and back surfaces of the corrective lens will reflect light. First of all, the reflected light is wasted so that the illumination of the eye (light source path) and the image of the eye (light to camera) will be attenuated. Second, the reflection from the illumination may be bounced back into the camera, which will be very bright and cause the auto-iris of the camera to produce a darker image of the eye.

If there is a problem with the front surface reflection of corrective lenses, try adjusting the angle of the lens relative to the camera-LED assembly. There are two ways to do this:

Slightly tilt the corrective lenses by moving the earpiece so that it is near the auditory canal (hole in the ear) rather than resting on top of the ear; this is fairly easy for lightweight springy metal frames, but may not stay in place with heavier frames.

Slightly move the camera so it is more than 45 degrees below the line-of-sight.



Arrington Research



When de-focusing, it is preferable to move the focal plane farther away from the camera, rather than closer to it, because the latter will sharply focus on any dirt or debris that is on eye-glass lenses.

5.4 Thresholding – All Systems

For simplicity and ease of use, keep the *AutoImage* and *Positive Lock Threshold* tracking options checked. If you do need to adjust these manually, refer to [Chapter 6](#).



The *PupilScan Area* must be at least 50% of the *EyeCamera* image area for the *AutoImage* to function well. Refer to [6.5](#)



If your subject has a small pupil then you need to scan more densely (scan dots closer together); adjust the Scan Density to 5.

5.5 Calibration

Refer to the subsection here below for your particular system.



Try to use at least 12 calibration points, 16 points usually provides very good calibration. Using 6 points will provide accurate calibration only along the horizontal axis and should ordinarily be avoided.



It is good practice to integrate slip-correction into your experiments. Choose a good center point and perform the slip correction at the working distance.

This section is the Quick Start description of calibration; more detailed information is provided in [Chapter 8](#).

5.5.1 Calibration (Head Fixed and HMD Systems)

For the *HeadLock™* System, *QuickClamp™*, *Remote* system, or other systems where you are fixing the head, with respect to the display monitor, on which the *Stimulus* window is to be displayed, such that when the subject is looking straight ahead, their Position of Gaze is approximately two thirds of the way up the monitor vertically and centered horizontally. The *Stimulus* window should be placed so that the subject can see it easily when positioned comfortably. This is best achieved using a second monitor and full screen stimulus display, refer to [Chapter 12](#).

After successful thresholding as outlined in [5.4](#) follow the steps below:

1. Warn the subject of the onset of the calibration stimuli to ensure successful calibration.
2. Instruct the subject to look directly at the center of each stimulus until it converges to a point. The default is for the calibration *StimulusPoints* to appear in random order.
3. Start the calibration by pressing the *Auto-Calibrate button* on the *EyeSpace* window. The warning message “*Get Ready*” will appear briefly on the screen to draw the subject’s attention to the start



Arrington Research

of the calibration process. This can be suppressed, or the display time adjusted, via the [EyeSpace window > Advanced button: Advanced Calibration window > Warning slider](#).

4. During the calibration process, ensure the pupil is accurately located at all times by monitoring the green dots and the yellow oval, i.e., monitoring the image segmentation.
5. Check the calibration by examining the positions of the calibration *DataPoints* in the *EyeSpace* window. Successful calibration will be indicated by a rectilinear and well-separated configuration of green dots corresponding to the locations of the pupil at the time of calibration point capture. See [Figure 14](#) and [Figure 15](#).
6. Stray calibration *DataPoints* can be identified and re-calibrated or omitted. The [EyeSpace window > DataPoint slider](#) allows the user to select stray calibration points to be recalibrated. The active *DataPoint* is highlighted in the *EyeSpace window > graphics well*. *DataPoints* can also be selected with the mouse by left clicking the calibration point in the graphics well.
7. Select the stray calibration point by left mouse clicking that *DataPoint* in the *EyeSpace window > graphics well*.
8. Instruct the subject to look at the center of the stimulus and represent the calibration point by pressing the [EyeSpace window > Re-present button](#). The warning message “*Get Ready*” will appear briefly on the screen at the calibration point location to draw the subject’s attention to the representation location. (This can be suppressed or the display time adjusted.) This exercise can be repeated with as many calibration *DataPoints* as necessary. If the calibration points are not rectilinear, for example, there are lines crossing then complete re-calibration is necessary.
9. If a particular point cannot be recalibrated, then select that point and press the [Omit button](#).
10. A quick check of calibration accuracy may be done by asking the subject to look at particular points on the stimulus and using the *GazeSpace* window to verifying that the gaze point matches up with the points looked at.

5.5.1.1 HMD Partial Binocular Overlap

Some wide field-of-view (FOV) head mounted displays (HMDs) employ partial binocular overlap (PBO) of the displays, such that there is stereo in the central overlapped area, but monocular imagery in the eccentric, temporal parts of the displays. This presents a challenge to calibration, because there are calibration points in each display that are not in the other display.

Additionally, PBO displays are often canted (tilted) such that they are not coplanar with one another. This further complicates things because the imagery, and the calibration points, must be keystone predistorted for them to align.

ViewPoint completely handles these situations by allowing separate, custom calibration sets for each of the two eyes. See sections [8.12.4.2](#) and [19.12.20](#). The calibration points displayed in the *GazeSpace* and *Stimulus* windows are now color coded according to the specified eye display color.

5.5.2 Calibration (*SceneCamera*)

Calibration is performed relative to the viewing area (pixels) of the *SceneCamera*, NOT the video image content. This is analogous to calibrating relative to the stimulus display screen and NOT the image displayed on it.



After ensuring successful eye image thresholding, perform the following calibration steps to provide Position of Gaze with respect to the *SceneCamera* image.

1. Select menu item: *Stimuli > View Source > Scene Camera*. This will cause the video from the *SceneCamera* to be displayed in the *GazeSpace* window. It also enables the calibration settings *Snap Presentation Mode* and *Auto-Increment* (refer to *EyeSpace* window advanced settings) and causes the calibration *StimulusPoint* array to be displayed in the *GazeSpace* window.
2. Adjust the *SceneCamera* so that the center of the subjects straight ahead field of view is at the center of the image.
3. If necessary, adjust the brightness and contrast of the *SceneCamera* image using the *Controls window > Scene tab > Brightness and Contrast sliders*. Reduce brightness such that overlays are visible.
4. Ask the subject to stand or sit comfortably and to remain still for the duration of the calibration process. **They should not look at the computer screen.**
5. Position a *StimulusPoint* (pen top, your finger tip, or a laser pointer dot on a wall or board) in front of the subject such that the *StimulusPoint* appears inside the *GazeSpace* window inside the active (blue) calibration circle. All calibration *StimulusPoints* should be in a plane that is perpendicular to the subject's line-of-sight.



For best accuracy, try to calibrate at about the viewing distance at which the subject will typically be working.

6. Ask the subject to move their eyes (keeping their head still so that the pen remains in the active calibration point) to look at the *StimulusPoint*.
7. When you are sure that they are looking at the *StimulusPoint* (get verbal confirmation) select the *EyeSpace window > Re-Present button*. The * (asterisk) on this button indicates "*Snap Presentation Mode*" and the ++ indicates "*Auto-increment*". The *F8 FKey* can be used as a shortcut key for this button (this is the default FKey command).
8. Repeat steps 5-7 with each active calibration point until you have completed the set.
9. Examine to the distribution pattern of the calibration *DataPoints* in the *EyeSpace* window. Successful calibration will be indicated by a rectilinear and well-separated configuration of green dots corresponding to the locations of the pupil at the time of calibration point capture. Uniform curvature of the field of dots is acceptable. See [Figure 14](#) and [Figure 15](#)
10. Stray calibration *DataPoints* can be identified and re-calibrated. Select the stray calibration point by left mouse clicking the *DataPoint* in the *EyeSpace* window. Alternatively the *EyeSpace window > DataPoint slider* allows the user to select stray calibration points to be recalibrated. The active *DataPoint* is highlighted in the graphics well.
11. Repeat steps 5-7 for the stray calibration point. This exercise can be repeated with as many calibration *DataPoints* as necessary. If the calibration points are not rectilinear, for example, and there are lines crossing, then complete re-calibration is necessary.
12. A quick check of calibration accuracy may be done by asking the subject to look at particular points on the stimulus and using the *GazeSpace* window to verifying that the *GazePoint* matches up with the points looked at.



Arrington Research

13. A consistent offset in position can be corrected using the slip-correction feature. Select a calibration *DataPoint* in the middle of the group. Repeat steps 5-7 above but press the *Slip-Correction button* instead of the *Re-present button*. This automatically adjusts the calibration to compensate for the measured slip in the X and Y planes.



The line-of-sight of the *SceneCamera* is not exactly the same as that of the eye. This will cause a vertical offset called parallax error, when the distance of the calibration point (plane) is different from the distance of the *GazePoint*. The Binocular version of the *SceneCamera* system includes automatic and manual correction for this parallax error. See section [9.2](#).

5.6 Stimuli

5.6.1 Choosing the Type of Stimulus

The *ViewPoint EyeTracker* can be used with a variety of types of visual stimulus environments, including (a) stimulus images presented in the *ViewPoint Stimulus* window, (b) real world environments as would be viewed while walking about captured by the EyeFrame *SceneCamera*, or (c) interactive work on a computer display captured as a movie of screen snapshots. See menu item: *Stimuli > View Source >*

You can also interface to third party applications e.g. stimulus presentation and experimental control programs, and virtual reality applications; however these are beyond the scope of this Quick Start Section. See the [~/ViewPoint/Interfaces/](#) folder.

The following sections describe setting up various stimulus environments and saving data in these environments.

5.6.2 *ViewPoint Stimulus* Window

Using *ViewPoint* in head fixed mode or with an HMD, you can present stimuli in the *ViewPoint Stimulus* window as follows:

1. Present static BMP images and sequences of BMP images in the *ViewPoint* stimulus window. The *GazePoint* over the images is recorded. Select menu item: *Stimuli > View Source > ViewPoint Stimulus window*.
2. Stimulus movies: Future capability.

5.6.3 Interactive Computer Display

Screen Movies: the subject's *GazePoint* is recorded as they interact with the computer display screen. A movie is recorded that can then be played back showing the eye movement as the subject scrolls, opens and closes windows, etc. Select menu item: *Stimuli > View Source > Interactive Computer Display*.

5.6.4 *SceneCamera* Systems

When the *SceneCamera* option is enabled, the user can select menu item: *Stimuli > View Source > Head Mounted SceneCamera* the stimulus is then the real world scene as the subject moves freely around. If the *SceneCamera* option is enabled, whenever a *ViewPoint DataFile* is recorded, a *SceneMovie* will also be created. There are several options for the *SceneMovie*, including format, selection of overlay graphics, and compression. *ViewPoint* synchronizes the *SceneMovie* with the *DataFile* by asynchronously inserting a TimeStamped frame number every time a movie frame is stored. This allows retrieving the



corresponding *SceneCamera* video frame image as the data from the *DataFile* is read. Eye movement data values can also be stored inside the movie frame (currently this is limited to uncompressed movies). This allows the eye tracking data to be later extracted from the *SceneMovie* without reference to the *DataFile*.

5.6.5 Show SceneVideo in *Stimulus* window

To display the *SceneCamera* video in the *Stimulus* window, check the *Controls window > Scene tab > Show SceneVideo in Stimulus window checkbox*, or use CLI: `ShowSceneVideoInStimulusWindow BoolValue`

5.7 Data Collection – All Systems

Now that the system is calibrated, data can be collected. You can select the required sampling rate using the monitor icon on the *EyeCamera* Window. This will bring up a menu with various options depending upon the product that you are using. Refer also to [Chapter 13](#) Data Visualization & Analysis.

To start recording the data to file select menu item: *File > Data > New Data File*. This will prompt you to create a new *DataFile* in the default folder: `~/ViewPoint/Data/`. When a *DataFile* is open the *Controls window > Record tab* will show two lines similar to this:

```
File: "2014-11-25;13-17-23.txt"  
Data: OPEN 00h 00m 0.773s      PAUSED
```

The menu item: *File > Data > Unique Data File* will create a new data file with a unique name to be opened without having to go through the File Dialog box.

Recording can be paused at any time by selecting menu item: *File > Data > Pause Data Capture ^P*. When data storage is paused the *Controls window > Record tab* will display **PAUSED**, as shown above. To stop recording, select menu item: *File > Data > Close Data File*. The window will indicate that the file is **CLOSED**.

You can also use the *Controls window: Record tab* for easy data collection controls.

See [Figure 8](#) below:

Figure 8. Controls Window: Record Tab



New Recording button: Creates a new *DataFile* with a unique name constructed from the date and time, and immediate starts recording (unless Pause is active). This button has the same effect as the GUI menu item: [File > Data > Unique Data File](#), or the CLI command `dataFile_NewUnique`. If the Shift-key is depressed when this button is pressed, the new *DataFile dialog box* will appear which allow the user to enter a *DataFile* name of their choosing; this is the same as CLI command: `dataFile_DialogBox`, or GUI menu item: [File > Data > New Data File](#).

Pause/Resume Button: Pressing this button toggles between pausing and resuming the recording of data into the *DataFile*. This button has the same effect as the GUI menu item: [File > Data > Pause Data Capture](#), or the CLI command `dataFile_Pause Toggle` (or the pair of CLI commands `dataFile_Pause` and `dataFile_Resume`).

Mark Button: Sequentially inserts an alphabetic marker character ‘A’ to ‘Z’ into the *DataFile*. The inserted character can be observed in the *Seconds* line graph in the *PenPlot* window. On the first call, this is the same as the CLI command: `dataFile_InsertMarker A`. On subsequent presses, this button has the same effect as the CLI command `dataFile_NextMarker`. The sequence restarts after it gets to ‘Z’. The sequence can be reset at any time by holding the *Shift-key* when depressing the *Mark button*, or with the CLI command: `dataFile_RestartMarkers`. See also GUI menu item: [Windows > KeyPad / DataMarker](#).



Close Button: Closes the *DataFile*. This button has the same effect as the GUI menu item: *File > Data > Close Data File*, or the CLI command `dataFile_Close`.

Browse Button: Opens the *DataFile open dialog box*, from which the user can select a data file that will be opened by the *DataAnalysis* program. This button has the same effect as the GUI menu item: *File > Data > Edit Data File ...*, or the CLI command: `dataFile_AnalyzeDialog`.

Analyze Button: Immediately closes any open data file and then opens the most recent dataFile in the *DataAnalysis* program. This button has the same effect as the GUI menu item: *File > Data > Analyze Last Data File* or CLI command `dataFile_CloseAndAnalyze`.

To specify a different Data Analysis program, refer to

5.8 Recording Scene and Screen Movies

5.8.1 Recorded Movie Format

The default movie format for most systems is AVI 2.0. We recommend this format be used unless there is some good reason to do otherwise. HighSpeed systems may use VPM movies for better performance. Recording length is limited only by disk space.

Menu item: *File > Data > SaveMovie Compression (AVI2 only) > ...* shows all codecs currently available on your computer in the menu list. CLI command: `saveMovie_DumpCodecs` prints the available codecs in the *History* window. Available *codecs* depend upon the system; third party codecs can be added.



Compression can be CPU intensive and may result in lost frames. Lossy compression will corrupt any data values that are embedded in the movie frames

Use Display of Movie Data

ViewPoint provides the user with two options for way the data is saved and displayed on the scene movie. Care must be taken to choose the one most suited to your needs. Menu Item: *File > Data > Save Movie DataMode > ...*.

1. Painted (default)

When this option is selected, all the overlay graphics selected to be shown in the *GazeSpace* window are also permanently painted into the frame images of the movie. This is a very convenient way to make easily distributable movies with gaze position overlays. These overlays cannot be removed. The user should take care to deselect any overlay graphics, such as the calibration array, if they are not wanted as a permanent part of the movie.

2. Embedded

When selected, several eye tracker data values are also stored inside the movie frame. This allows the eye tracking data to be later extracted from the SceneMovie without reference to the *DataFile*, and it allows the user to choose which data to overlay during the post-hoc analysis. Currently this is only available with no compression.

5.8.2 Display of Overlay Graphics in Recorded Movie

Historically, the *SceneCamera* movie was recorded with the same overlays that were selected for the *GazeSpace / SceneCamera* window. This often led to movies with unwanted overlays, because people forgot to de-select options like *Calib Region* and *ROI Regions*. When the *SceneCamera* is active, a third column of check boxes will appear in the Controls window > Display tab, which allow control of what overlays appear in the *SceneCamera* movie (*SceneMovie*). These can also be separately set with CLI command: `SceneMovieGraphicsOptions +option -option`, as well as for the *GazeSpace* window, with the previous command: `GazeSpaceGraphicsOptions +option -option`.

5.8.3 Compression

When recording AVI movies, the user can select to compress the movies if required. Select menu item: *File > Data > SaveMovie Compression*. The default is None, that is no compression, because compression increases the CPU burden, can result in loss of data because of the extra time required, and if not lossless, will corrupt any data that is embedded inside the movie frames. The codecs that appear depend upon which version of the operating system that you are using, and which codes the user has downloaded or installed with other applications. Be careful, not all codecs are free.

If available on your computer, we have found reasonable success using *Intel Indeo version 5.10* and higher codecs (*IV50* is the fourcc code). This is suggested because it is supposed to be available on most all machines (relieving the user from the ordeal of selecting a suitable codec, searching for and installing codecs and getting involved with cumbersome codec licensing issues), and because it provides reasonable quality images without overwhelming CPU burden.



Some compression codecs are time consuming and may slow down the performance of *ViewPoint* and its ability to save data and movie frames.



Digital movie creation and playing is not a fully mature area in computer science; as it evolves, so will with *ViewPoint EyeTracker* movie interface and format and compression options.

5.9 DataAnalysis program

AVI movies with or without Painted Data, either Compressed or Uncompressed, will be loaded automatically into the *DataAnalysis.exe* program (see separate PDF documentation) when the corresponding data file is loaded.



The *ViewPoint DataAnalysis.exe* program is currently provided as a beta version, without charge and AS IS; it may be useful, but ARI currently provides no official support.

5.10 Analysis Options

5.10.1 Head Fixed and HMD Systems

There are many options to view and analyze the data recorded, including:

1. Real-time in the *GazeSpace* and *PenPlot* Windows.
2. Using the *DataAnalysis* program provided free, as is, with *ViewPoint*. See separate PDF documentation.



3. Post-hoc analysis of the ASCII data files using *Excel*, *MATLAB*, *Mathematica*, etc.

5.10.2 Playing Scene and Screen Movies

AVI without Painted data requires the associated *ViewPoint DataFile* for the eye tracking data. The movie frame number is interleaved with the data records in the *DataFile*.

AVI movies with Painted Data, either Compressed or Uncompressed, can be easily distributed, downloaded, and played with most any movie player software, including Microsoft Media Player. This is by far the simplest and easiest, and is therefore the default.

To play VPM movies you need to use the `VPM_Viewer.exe` program. This extracts the eye tracking data from the frames of the movie and provides the user with data display options.

5.11 Frequently Used Settings

You can create *Settings* files and place them in the `/ViewPoint/Settings/StartUp/` folder to specify frequently used settings that you want done when *ViewPoint* starts. For example, you may wish load a particular image. When *ViewPoint* is launched it loads in all of the text (`*.txt`) files in this folder, which can reduce setup time.

5.12 Preferred Window Layout

A preferred startup window layout can be saved using menu item: *File > Settings > Save Window Layout*. This can then be reloaded using the load settings menu item. Alternatively, place it in the folder: `~/ViewPoint/Settings/StartUp/`.

5.13 Accelerator Keys & FKeys

Accelerator keys are used to make menu selections with the keyboard, rather than the mouse. The most current list can always be found within *ViewPoint* by selecting menu item: *Help > Info > ShortCuts tab*. The circumflex character '`^`' represents the *Control-key* held down as a *modifier-key*.

The user can associate an *FKey* with a *CLI* action. This is done via the *CLI* interface (see section [16.5](#)); for example:

```
FKey_cmd 11 { dataFile_NewUnique }
FKey_cmd 12 { dataFile_Pause Toggle }
FKey_cmd 1 { videoFreezeSync Toggle }
// Can use Toggle argument instead of On/Off, allowing one FKey to be used!
```

These associations can be viewed in the menu: *Help > Info window > ShortCuts tab*, refer to [19.26](#)

5.14 Printing

Many of the *ViewPoint* windows can be printed using menu item: *File > Print > ...* To include the current date and time on the prints, check menu item: *File > Print > DateTimeStamp Printouts*.



You may want to select Freeze before Print to prevent pull down menu occlusion.

Chapter 6. Locating the Pupil and Glint (All Systems)



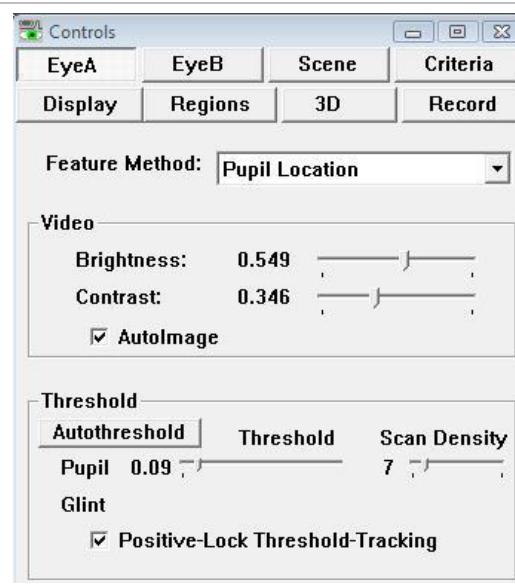
The *ViewPoint* software comes with two very powerful features that will automatically and continually provide the best setting for tracking each subject:

- ④ **Video Autolmage:** Click the *Controls Window > Eye tab > Autolmage check box*. When checked *ViewPoint* will automatically adjust the brightness and contrast values to optimal settings. Only the region within the pupil scan area is examined, so the *pupil scan area rectangle* must be of sufficient size for the algorithm to sample a range of gray levels, otherwise the algorithm will fail.
- ④ **Positive-Lock Threshold-Tracking:** Click the *Controls Window > Eye tab > Positive-Lock Threshold-Tracking check box*. Positive Lock provides continuous automatic feature threshold adjustment. Note: this only adjusts the pupil threshold; the glint threshold must be adjusted manually.

It is highly recommended that the user works first with these features before making any manual adjustments. However, there will occasionally be situations and subjects that may require some manual intervention to provide the most successful tracking.

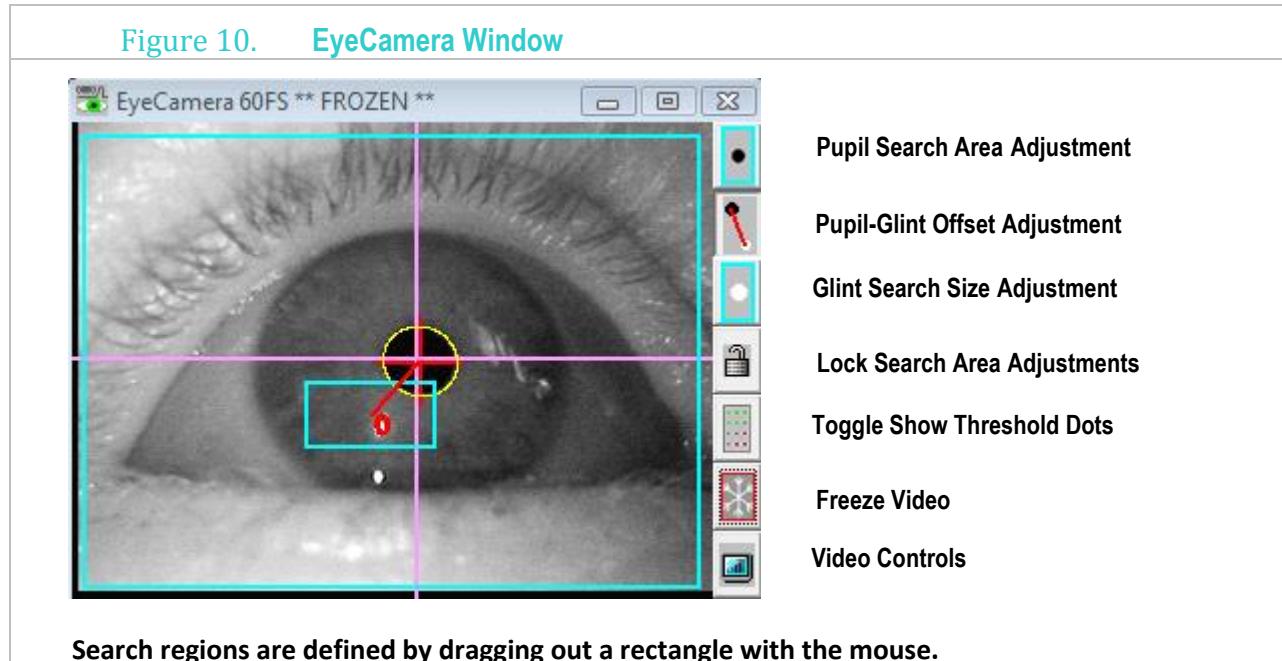
The *Controls window: EyeImage tab* is shown below

Figure 9. Controls Window: EyeA or B Tab



6.1 EyeCamera Window

The *EyeCamera* window displays the video-image of the eye, as well as overlay-graphics that graphically provide information about image segmentation and performance, see [Figure 10](#). The overlay graphics include: Thresholding results (e.g., green dots indicating dark areas), Pupil Location and Diameter Calculation results (yellow OvalFit to pupil), Corneal Reflection location results (red OvalFit to glint). The mouse is used in this window to pull (drag-out) a rectangle to define a limited search area, sometimes called a gate, for both the Pupil and Glint.



6.2 Feature Method

A feature is any content in the video image that can be identified, located and tracked; these include the dark pupil and the bright corneal reflection, i.e., glint. The [Controls window > Feature Method combo box](#) allows the user to select which features to use. The feature methods fall into two basic categories: single DataPoint methods and multiple DataPoint methods. Which method to use is probably best determined through experimentation.

6.2.1 Single DataPoint

The single DataPoint methods, for example using [Pupil Location Only](#) or specular corneal [Glint Location Only](#) are very sensitive to slight sideways head movements. Any head movement is completely confounded with eye movement. The Glint Location method is normally useless, but is included for educational purposes.

6.2.2 Multiple DataPoint

The multiple DataPoint method uses the [Glint-Pupil Vector](#) difference between (i) the center of the pupil and (ii) the center of the specular corneal reflection.

6.2.2.1 Advantages

This is more robust against small movements of the head with respect to the camera. The corneal reflection (CR) and the dark pupil (DP) move together as the head moves (translates in the x-y plane that is normal to the optical axis of the camera). By taking the vector difference between these two signals, a relatively translation-invariant point-of-regard eye tracking system can be achieved.

6.2.2.2 Disadvantages

There are several disadvantages of the *Glint-Pupil Vector* method. (a) There are now two sources of video and segmentation noise instead of one. (b) Given a change in viewing direction, the vector variation is smaller than the variation of the pupil (or the glint) alone. The result is a lower resolution and lower signal to noise ratio. (c) The vector method is sensitive to a different type of translation error. The *Glint Location Only* and *Pupil Location Only* methods are particularly sensitive to translation of the head in a horizontal (sideways, x-axis) or vertical (up/down, y-axis) direction and less sensitive to in-and-out (closer or farther from the camera, z-axis) movement of the head. By contrast, the *Glint-Pupil Vector* method is robust against x-axis or y-axis movement, but is more sensitive to z-axis movement of the head, because this affects the length of the calculated vector; that is, the vector becomes shorter as the head is moved backward away from the camera. This is particularly true when the camera is close to the eye, because the angular field of view is wider. If a more remote camera is used it will be less sensitive to Z-axis variation.

6.2.3 Slip Compensation

The Feature Method combo box also includes a *Slip Compensation* method described in section [6.9](#), and data Simulation options described in section [6.3](#) below.

6.3 Simulation of Gaze

Simulated data can be very useful for learning how things work and for debugging. Select the *Controls window > Regions tab > Simulation radio button*. The adjacent drop-down menu allows selection between simulation modes: *Manual* or *Pattern* that are described here below.

6.3.1 Manual Simulation

Next to the *Simulation radio button*, select *Manual* from the drop down menu. Manual Simulation allows the user to use the mouse to simulate the POG. Click or hold and drag the mouse button in the *GazeSpace* window. Note that the smoothing operation is still applied, so unless smoothing is set to one (turned off) it may take several clicks at a location before the gaze point indicator moves to the location of the mouse. Holding and dragging a left mouse-click will move EyeA's POG. Holding and dragging a right mouse-click will move EyeB's POG. To move both eyes together, you must start by holding down one of the mouse-clicks, and then engage the other mouse-click before moving the mouse. Now both eyes should be locked together and both POG should move together effectively creating a single point. The pupil and glint quality codes are set to best-quality (as soon as the user moves the mouse in the *GazeSpace* window) so that subsequent operations are not impeded. *Note: the eye video will be frozen.*

6.3.2 Pattern Simulation

Next to the *Simulation radio button*, select *Pattern* from the drop down menu. This provides simulated eye positions data in the form of a continuous test pattern that is useful for developing and debugging interfaces, such as layered SDK program interfaces, Ethernet client/server connections, etc. See CLI: [controlsTab Regions](#); [gazeSpace_MouseAction Simulation](#); [simulationMode Pattern](#)

[GazeSpace_MouseAction None](#)

6.4 Thresholding

The software attempts to locate the pupil by searching for a dark region within the pupil search area. Select: *Controls window: EyeA or EyeB tab, Feature Method group, Pupil Location*. Adjusting the *Controls window > Threshold group Pupil slider* controls which luminance values to include or exclude. The slider adjusts the threshold level (sensitivity) for the pupil. Moving the slider to the right raises the dark pupil threshold ceiling, allowing more (lighter) gray levels to be counted as part of the dark pupil. Clear focus is not always optimal for obtaining good segmentation, because display screen reflections over the pupil can sometimes confuse the image segmentation process. Also, defocusing can cause the specular corneal glint to appear larger, which can make it easier to locate.

6.5 Setting the Scan Density

The resolution at which the program samples pixels in the video image is adjusted by the *Controls window > Threshold group Scan Density slider*. The finest resolution is with the slider set to the left, as the slider is moved to the right, only every n'th pixel is examined, where n is the actual Scan Density value.

ViewPoint works by isolating the pupil and corneal reflection in the video image. To segment the image properly, the intensity-threshold levels must be appropriately set. The pupil and the corneal reflection are located by first taking the mean position of all sample points within threshold limits. The spatial resolution of the sampling may need to be adjusted to optimize speed or accuracy. Moving the slider to the right increases the sample spacing (coarse) which reduces the sampling resolution and correspondingly the number of dots shown. Moving the slider to the left increases the resolution (fine). The result of the sampling resolution is graphically displayed when the *Show Threshold Dots button* on the *EyeCamera* window is toggled on.



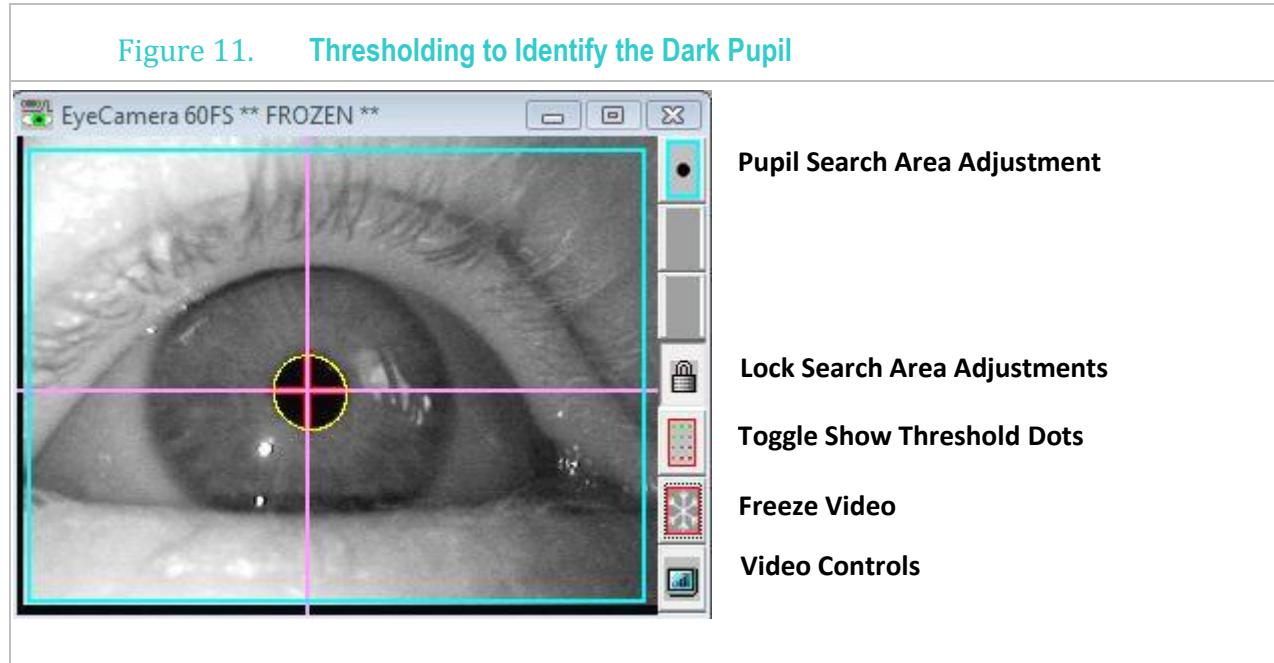
It takes time to scan pixel values and to paint them. It is very useful for determining what Segmentation Threshold and Scan Resolution settings are optimal. Once optimal settings have been found, the scan density should be reduced as far as possible to reduce the computational burden. If fine scan resolution and a large scan area are required, then the Show Threshold Dots may be turned off to remove the computational burden of painting the colored dots.



If your subject has a small pupil then you need to scan more densely (scan dots closer together); adjust the Scan Density to 5. For even smaller pupils you will need to change the minimum allowed with the CLI: [minimumPupilScanDensity 1](#).

6.6 Manual Thresholding of the Dark Pupil

Follow the steps below to manually undertake the manual Thresholding process for the Dark Pupil:



1. Toggle show threshold dots “ON” using the button in the *EyeCamera* window.
2. Ask the subject to fixate at the center of the display screen. If self-testing move the *EyeCamera* window to the center of the display screen.
3. Select the *Pupil Search Area Adjustment* icon at the top right of the *EyeCamera* window, illustrated in [Figure 10](#). Use the mouse to drag out a rectangle that limits the area in which to search for the pupil. In particular, use this to eliminate dark shadow areas that could be confused with the pupil. The *OvalFit* algorithm will continue to fit the pupil beyond the limits of the pupil search area. Consequently, the pupil search area can be substantially smaller than first imagined.
4. Press the *Controls window > Threshold group > AutoThreshold button*.



Pressing the *AutoThreshold button* automatically sets desirable Light-Reflection and Dark Pupil threshold levels. After this is done, the user may want to make further adjustments, which is easily done using the sliders.

5. If necessary, adjust the *Controls window > Pupil Threshold slider* to ensure that the green dots appear only in the pupil and that the yellow oval outlines the pupil and is fairly circular (note: that the *PenPlot window > Pupil Aspect ratio graph* displays the oval’s aspect ratio, 1 = perfect circle). The yellow ellipse indicates where the program has located the area of the pupil.
6. If too many dark areas other than the pupil have green dots, adjust the dark pupil threshold slider to the left. Alternatively, if insufficient pupil area is being identified as dark (with green dots), adjust the *Pupil Threshold slider* to the right. The *Positive-Lock Threshold-Tracking* attempts to provide optimal tracking; however it is not appropriate for all subjects or situations.



Arrington Research

7. After the pupil has been isolated, adjust the scan density to use the minimum number of dots that can reliably and consistently locate the pupil. Correct thresholding of the Dark Pupil is illustrated in [Figure 10](#). The default set on startup is optimal for most situations and you should very rarely need to adjust this.



Unnecessarily high scan density settings together with large scan areas can cause the frame rate to drop and data can be lost.

8. Ask the subject to look at each of the four corners of the *Stimulus* window to ensure that the pupil remains in the search box and to ensure accurate thresholding for all potential eye movements. i.e. the yellow oval outlines the pupil and is fairly circular. If self-testing, move the eye camera around to the four corners of the display to view the effects of pupil segmentation at different Position of Gaze. At any point the experimenter can toggle the image display *On/Off*. When *Off* it provides a still image of the eye to aid identification of successful thresholding. This is accomplished by freezing the video by pressing the *Freeze Video* icon button at the bottom right of the EyeCamera window, or by pressing the *F1* function key that has its default CLI command *videoFreezeSync Toggle*. When frozen, the icon button will be outlined in red and a check mark appears next to the menu item. The EyeCamera window will also indicate **** FROZEN **** in the title bar.

Selecting the option for *Positive-Lock Threshold-Tracking* will help in many situations.



It takes time to paint the threshold dots on the screen, but it is very useful for determining what Segmentation Threshold and Scan Resolution settings are optimal. Once optimal settings have been found, the Toggle show threshold dots may be turned off to remove the computational burden of painting the colored dots.



This section describes pupil only thresholding. With the Pupil Only method, any X, Y plane head movement will be confounded with eye movement. To measure movement that is invariant to X, Y plane head movement, use the glint-pupil vector method.

6.7 Step-by-step guide for *Glint-Pupil Vector* method

Follow the steps below to manually undertake the thresholding process for the Glint-Pupil vector method:

1. Select *Controls window > Glint-Pupil Vector*.
2. Follow the steps in [6.6](#) to threshold the Dark Pupil.
3. After the pupil has been isolated, adjust the scan density to use the minimum number of dots that can reliably and consistently locate the pupil. Correct thresholding of the Dark Pupil is illustrated in [Figure 11](#).
4. Ask the subject to look at each of the four corners of the *Stimulus* window to ensure that the pupil remains in the search box and to ensure accurate thresholding for all potential eye movements, i.e. the yellow oval outlines the pupil and is fairly circular. If self- testing move the eye camera around to the four corners of the display to view the effects of pupil segmentation at different Position of Gaze.
5. Select the Pupil-Glint Offset Adjustment icon on the *EyeCamera* window, illustrated in [Figure 10](#) Use the mouse to drag out the offset vector from the center of the pupil to the center of the corneal glint.

6. Press the button to select the glint search size adjustment mode, and then use the mouse in the *EyeCamera* window to drag out smallest rectangle that catches the glint at all possible eye positions. Because the glint search size moves relative to the calculated center of the pupil, the absolute placement of the glint search size specification rectangle with the mouse is not important, only its size is important.
7. After the glint has been isolated, adjust the scan density to use the minimum number of dots that can reliably and consistently locate the glint. Correct thresholding of the glint is illustrated in [Figure 10](#)
8. Ask the subject to look at each of the four corners of the *Stimulus* window to ensure that the glint and pupil both remain in the respective search boxes and to ensure accurate thresholding for all potential eye movements, i.e. the yellow oval outlines the pupil and is fairly circular and the red oval outlines ovals (red and green for dual glints) outline the glint and is fairly circular. If self- testing move the eye camera around to the four corners of the display to view the effects of pupil and glint segmentation at different Position of Gaze.

At any point the experimenter can toggle the image display *On/Off*. When *Off* it provides a still image of the eye to aid identification of successful thresholding. This is accomplished by pressing the *Freeze Video* icon button at the bottom right of the *EyeCamera* window. When frozen, the icon button will be outlined in red and a check mark appears next to the menu item. The *EyeCamera* window will also indicate "***** FROZEN *****" in the title bar.

Scan adjustments can be locked by selecting the *EyeCamera window > Lock Search Area Adjustments* icon (padlock with hasp closed).

6.8 Noise

The major problem that the user will face is to increase the signal to noise ratio. In the *Glint-Pupil Vector* mode, the basic measure of signal is the length of the vector from the center of the pupil to the center of the glint, so there are now two sources of noise. Noise comes from many things, for example: eyelid droop, eye blinks, extreme GazePoint angles that produce a reflection from the less smooth sclera (white part of the eye) and margins where the cornean curvature shows inflection, shadows, extraneous specular reflections, as well as the internal electrical video noise.

The first most obvious way to increase the signal is to increase the image size (move the camera closer or zoom in), so the pupil to corneal reflection difference vector appears larger in the *EyeCamera* window. The trade-off here is that smaller head movements can move the eye out of the view of the camera. The mapping function is calculated based on the calibration data, and it is only as good as the calibration. There are many sources of non-linearity. Consequently, mapping the raw eye-image-feature data to direction-of-gaze requires a sufficiently sophisticated non-linear mapping function. Obviously, if the subject is not looking at the calibration point when the data is sampled, then the calibration function that is calculated is not going to produce accurate GazePoint information.

6.9 Automatic Slip Compensation

Some hardware configurations, for example Head Mounted Displays (HMDs) can pose a problem when the camera position constantly slips with respect to the eye. Automatic Slip Compensation

combines the advantages of the larger signal space and reduced signal noise of the Pupil Location method, together with the translation-error robustness of the Glint-Pupil Vector method.

Select [Controls window > Eye tab > Feature Method > SlipCompensation](#) from the pull down menu.

Three parameters can be adjusted to specify the degree of compensation applied to the data using CLI commands. Refer to [19.9](#)

Note: do not confuse this with Calibration Slip Correction.

6.10 Feature Criteria

6.10.1 Pupil Aspect Criterion

The program can reject a dark segmented area as being the pupil based on its ability to fit a circle to that area. If the ratio of the minor-axis to the major-axis is less than the criterion level, then that area is rejected. If you are going to use this feature, then typically a criteria level of 0.6 is a good place to start. Enable the [PenPlot window > Pupil Aspect Ratio line graph](#) in the to graphically view the criterion level relative to the pupil aspect ratio data-value in real-time. Adjust the [Controls window > Criteria tab > Pupil Aspect Criterion slider](#) so that the threshold bar is below the data value for all potential eye movements, but above that for blinks. The pupil OvalFit changes color from yellow to orange when criterion is violated.

The variation of aspect ratio over the range of eye movements will depend on the viewing angle of the camera. The eye image in [Figure 10](#) was taken with a micro camera (arranged as in [Table 19 Schematic of the ViewPoint EyeTracker® System](#)). The pupil will appear more oval as the angle increases between the optical axis of the camera and the line-of-sight of the eye.

6.10.2 Width Criteria

ViewPoint also provides Minimum Pupil Width and Maximum Pupil Width criteria. Enable the [PenPlot window > Pupil Width line graph](#) to graphically view these criteria levels relative to the pupil width data-value in real-time. Adjust the [Controls window > Criteria tab > Maximum Pupil Width slider](#) and the [Minimum Pupil Width slider](#) so that the data value is between the threshold bars for all potential eye movements.

6.11 Alternative Segmentation Methods

Various algorithms are available for identifying the pupil and the glint (refer to [section 19.8](#)) center. Lighting conditions and performance considerations will determine which method is best for a particular job.

The [EyeCamera window > Monitor icon > Pupil Segmentation Method > ... pull down menu](#) is used to make a selection.

6.11.1 Ellipse

This fit provides a general rotated ellipse to the pupil, so that a good fit is made to oblique pupil images. It requires more CPU time than the Oval Fit method, but in some situations it can provide a more accurate calculation for the pupil center. Because there are more degrees of freedom, the ellipse may appear more wobbly than with the [Oval Fit](#) method, however the pupil center calculations is usually more accurate, because more points are used for fitting.

6.11.2 Centroid

Centroid means “Center of Mass”. This is a simple method that may be useful if there is difficulty discriminating the edge of the pupil, or if the pupil is very small.

The pupil location is at the average position of all points above threshold, weighted according to how much above threshold. Points below threshold are weighted more if they are darker. This centroid location is used as the starting point for its additional processing by the more sophisticated methods discussed in the next sections. No aspect ratio information is calculated.

6.11.3 Oval Fit

This method scans the area around the Centroid for extreme left, right, top and bottom values that are used as the coordinates of an unrotated (flat) bounding rectangle. The center of this bounding rectangle is taken as the center of the pupil. This method is more robust than the centroid method alone, and it takes less CPU time than the general rotated Ellipse method discussed earlier.

6.11.4 PupilScanArea Shape Options

The PupilScanArea is defined by a rectangular bounding box, but by default the actual scan area is an ellipse that is fit to this bounding box. This becomes apparent when the threshold dots are displayed. The user can change the scan area for the pupil to either rectangular or elliptical using CLI commands but we do not recommend it, because an elliptical scan area helps eliminate dark spots at the corners of the rectangle, which the software may interpret as a pupil.

6.11.5 Glint Segmentation Methods

The previous sections have focused segmentation and identification methods for the pupil. It is also possible to change the default segmentation method for the glint, though it is not normally required. The [*EyeCamera window > Monitor icon > Glint Segmentation Method > ... pull down menu*](#) is used to make a selection.

6.11.6 Edge Trace (only on special versions)

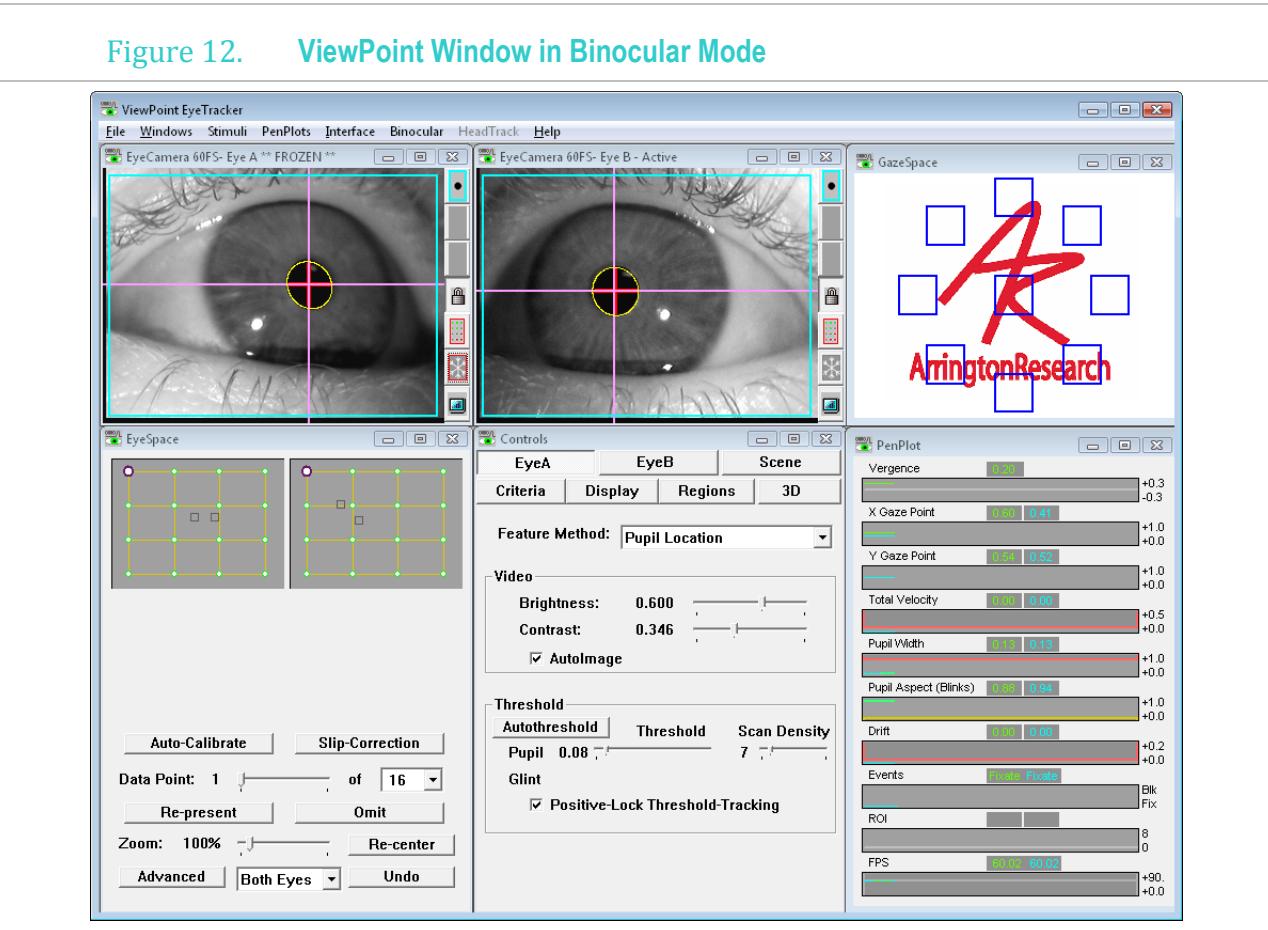
The pupil location is the center of the extreme values obtained during an edge trace around the pupil, starting from the point at 3 o'clock from the weighted centroid. This algorithm first scans rightward to find the right edge of the pupil. Next the algorithm traces the edge of the pupil, using the dark pupil threshold limit as the edge criterion. The extreme positions obtained during the edge trace are used to fit an oval, the center of which is taken to be the pupil location.

Chapter 7. Binocular Option

This section describes those features that are particular to the *ViewPoint EyeTracker®* binocular option.

7.1 To switch Operating in Binocular Mode

To switch from monocular to binocular operation, select menu item: *Binocular > Binocular Mode (toggle checkmark)*. See [Figure 12](#) below for the binocular mode window layout.



Alternatively, use the CLI command: **Binocular_Mode On**

This command may be placed in the *Settings* file startup.txt to automatically set your binocular preference when *ViewPoint* is launched. Refer to [Chapter 15](#).

7.2 CLI Prefix - EyeTarget Specifier

CLI commands can be targeted to specific eyes by using the *EyeTarget Prefix*, **EyeA:** or **EyeB:**, with no spaces between, for example: **EyeB:videoAutolImage ON**; see section [18.15](#).

7.3 Setup

Binocular mode requires about twice the processing time. For optimal performance in the binocular mode on slower machines, follow these suggestions:

- ④ The video modes may be different between Eye-A and Eye-B if required.
- ④ Set the video modes to High Precision rather than High Speed, unless the higher speed is actually required.
- ④ When collecting data, avoid moving the mouse, especially to resize or move windows, which may cause video frame loss especially on slower computers.

When Binocular Mode is selected, the *EyeSpace* window has an additional drop down box to specify which eye the calibration process is to apply to. See [Figure 12](#).

7.4 Storing Data

The data file will automatically have data columns appended for binocular mode. Quality markers are recorded for each eye which will allow collection of data from one eye, even if the other eye data has been rejected for some reason. The data file record format is typically a sequence as follows:

`Tag#, EyeA_data, EyeB_data, Count, Markers`

Refer to [Chapter 13](#) for details of data file format.

7.5 Real-Time Display of Binocular Data

The binocular data may be combined in a “cyclopean” average position during the display in the *GazeSpace* and *Stimulus* windows and in the *PenPlot* window:

See menu items:

Binocular > Show both eye positions

Both eye positions will be displayed as separate points in the *GazeSpace* and *Stimulus* windows.

Binocular > Show averaged Y-Positions

Two positions of gaze will be displayed in the *GazeSpace* and *Stimulus* windows. These will reflect the average of the vertical (Y) positions of both eyes.

Binocular > Show average of Eye Positions

One Position of Gaze will be displayed in the *GazeSpace* and *Stimulus* windows. This will take the average of the eye positions.

Parallax Correction (for SceneCamera systems)

Uses vergence to correct for vertical Parallax errors.

Note: Regions of Interest (ROI) hit lists are triggered by the (possibly smoothed & corrected) individual positions of gaze, but with no binocular averaging.

Note: The Binocular averaging option only affects the *GazeSpace* and *Stimulus* window displays.

Alternatively, use CLI: `binocular_Averaging [Off, only_Y, both_XY, ParallaxCorrection]`

This command may be placed in the *Settings* file startup.txt to automatically set your binocular preference when *ViewPoint* is launched. Refer to [Chapter 15](#)

Chapter 8. Calibration

This section describes the calibration process in more detail and also describes many of the advanced calibration features available to the user.



Before calibration, the pupil (and, if being used, the corneal reflection) must have been isolated with appropriate threshold settings.

ViewPoint starts up in a **coarsely calibrated** state that provides precise timing of raw (uncalibrated) eye movements. This is sufficient for many applications that can utilize relative eye movements, such as quadrant-wise “preference of looking” tasks. If your application requires more precise gaze point information, then further calibration will be required. Raw pupil and corneal reflection locations do not indicate where the subject’s position-of-gaze is. *Calibration Stimulus Points* are presented to the subject in the *Stimulus* window (*GazeSpace*) and corresponding eye feature locations in *EyeSpace* are saved as *Calibration Data Points*.

These raw DataPoints in *EyeSpace* (i.e. the EyeCamera video space) must be mathematically mapped to the subject’s *GazeSpace* (i.e. the visual-stimulus space) to provide a *calculated position of gaze* (POG). Calibration related CLI are described in sections: [19.12](#) and [21.8](#).

8.1 Calibration Carryover

There are substantial similarities between the eyes of different people, so it is sometimes possible to calibrate the system to one person, who is easy to calibrate, and then obtain reasonable data using that calibration for another person – though you will probably at least want to do a single point Slip Correction (see § [8.9](#)).

When using the *Glint-Pupil Vector* method you may want to obtain separate calibrations for each individual, because of individual variations in corneal curvature.

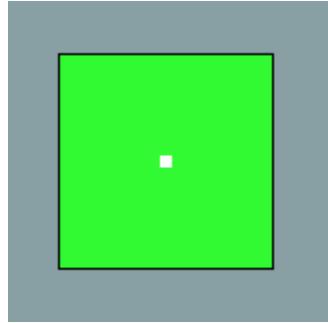
8.2 Choosing the Number of Calibration Points

The number of calibration points is selected by the user via the pull-down menu item in the *EyeSpace* window. The number of calibration points may be set to: 6, 9, 12, 16, 20, 25, 30, 36, 42, 49, 56, 64 or 72. A higher number of points may help with subjects that have corneal abnormalities or difficulty foveating. A setting of 12 or 16 is usually quite adequate. With fewer calibration points, good calibration accuracy is essential for each point. When a large number of points are used, the effect of any single point will be less. In general try to use at least 9 points to obtain a good calibration.

8.3 Automatic Calibration (Head Fixed)

Calibration stimuli are presented to the subject in the *Stimulus* window and also indicated to the user in the *GazeSpace* window. The subject should be instructed to foveate each point in turn. The calibration mode is “Tunnel Motion” calibration, where shrinking motion of a rectangular frame captures the subject’s visual attention and smooth pursuit brings the subject’s gaze point to each of the desired calibration spots in turn.

Figure 13. Calibration Stimuli



Calibration is started by pressing the *Auto-Calibrate* button in the *EyeSpace* window. If menu item: *Stimuli > Stimulus Window Properties > AutoShow on Calibrate* is also selected, then the *Stimulus* window will automatically be displayed full screen on the primary monitor. The option should be turned Off when assigning the *Stimulus* window on a secondary monitor.

The message “Get Ready” will appear briefly on the screen to draw the subject’s attention to the start of the calibration process. This can be suppressed or the display time adjusted via the *Advanced* section in the *EyeSpace* window. Speed of presentation of the calibration StimulusPoints can be adjusted via the *Advanced* section in the *EyeSpace* window.

The automatic calibration sequence may be stopped by pressing the *STOP Calibration* button in the *EyeSpace* window. Pressing the *ESC-key* will both stop the calibration and remove the full screen display if it is on the primary monitor.

For auto-calibration, it is usually preferable to randomize the presentation order of the calibration points, which is the default setting. With *Sequential Presentation Order* of calibration StimulusPoints, a leading source of calibration error is that the subject anticipates the presentation location of the next point, before the current StimulusPoint has finished. Explain to the subject that it is important to fixate on the calibration StimulusPoint until the point has completely disappeared.

The calibration stimulus presentation order type may be changed. Press the Advanced button in the *EyeSpace* window for access to the controls. See [section 8.12](#) for details on timing, presentation or, calibration stimulus type and color, etc.

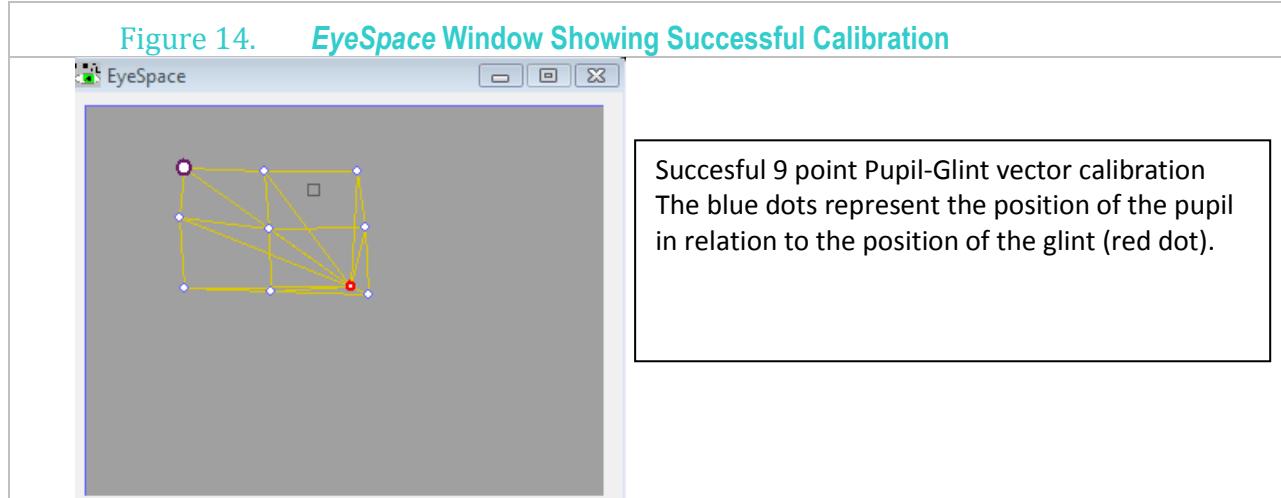
8.4 Assessing Calibration Success

A quick check of calibration accuracy may be done by asking the subject to look at particular points on the stimulus and using the *GazeSpace* window to verify that the gaze point matches up with the points looked at.

The arrangement of calibration DataPoints in the *EyeSpace* window provides a method of assessing how good the calibration data is.

Successful calibration will be indicated by a relatively rectilinear and well separated configuration of dots. The *Feature Method* (selected in the *Controls* Window, see [Figure 9](#)) determines how these DataPoints are plotted. If *Pupil Location* is selected, then the plot shows green dots corresponding to the locations of the pupil at the time of calibration point capture. The dots are joined by yellow lines that indicate the spatial relationship between the dots. If *Glint-Pupil Vector* is selected, the plot shows blue

dots corresponding to the locations of the pupil at the time of calibration point capture, but now they are shifted so that they are all relative to the corneal glint (red dot) that is plotted at the center of the DataPoint chart. The dots are joined by yellow lines that indicate the calibration index number (which is the order in which they are presented if sequential presentation order is selected).



The top part of the window shows a graphicsWell that represents the coordinate space of the *EyeCamera* window. In monocular mode the graphicsWell is 320x240, in binocular mode two smaller graphicsWells are shown, one for each eye. The center of the *EyeCamera* window is indicated in the graphicsWell by the intersection of the X and Y axes lines.

If *Pupil Location* or *Glint Location* modes are selected, a semi-transparent eye-colored circle shows the current pupil location, or the current glint location, respectively. In *Glint-Pupil Vector* mode the colored circle shows the vector difference between the pupil and glint locations, with the glint end of the vector fixed at the red dot.

For ease of DataPoint viewing, the calibration DataPoints and real-time feature points (pupil and / or glint) can be zoomed in or out using the *Zoom* slider, and can easily be moved by dragging with the *right mouse* button. The *Re-center* button repositions the DataPoints in the center of the graphics well.

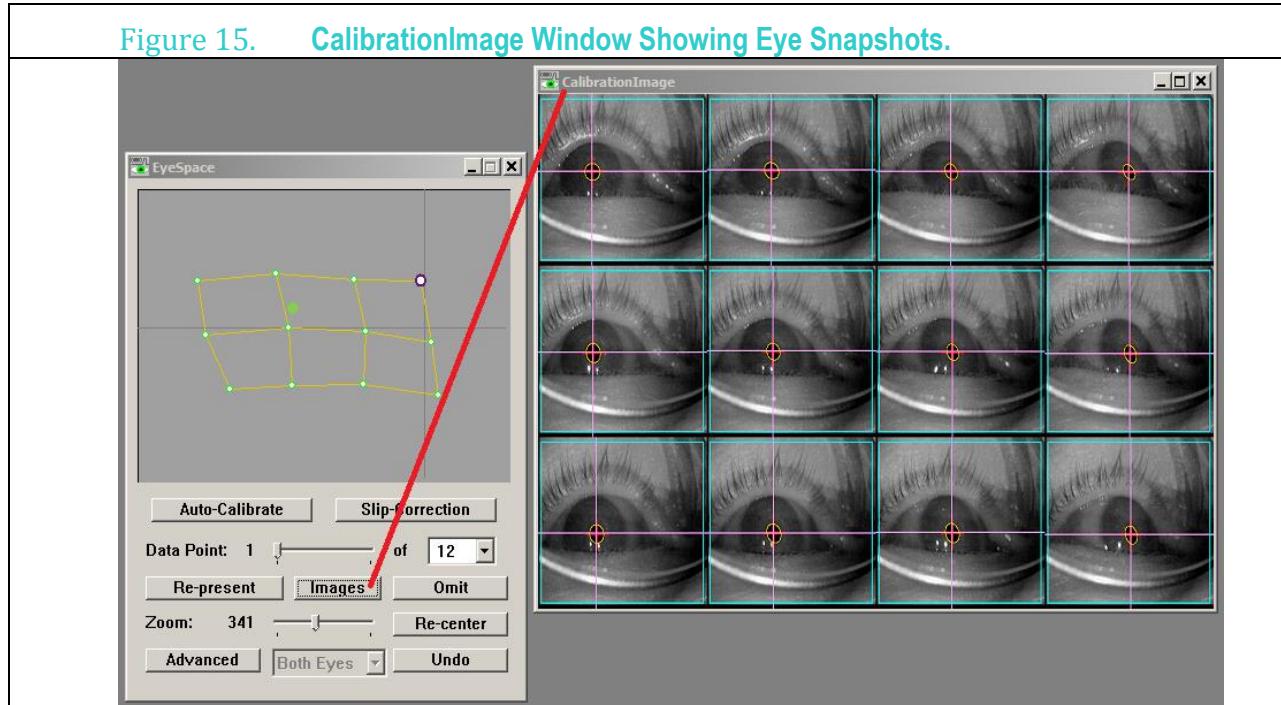
8.5 CalibrationImage

The *EyeSpace window > Images button* will show/hide the *CalibrationImage* window that shows the picture of the eye at the time when the calibration point was snapped, for each calibration stimulus point. This visually shows if the subject blinked during calibration, if glare is occluding the pupil, if a glint was lost, etc. This is very useful for troubleshooting.

This window initially shows a matrix of eye camera images that corresponds to the matrix of *calibrationStimulusPoints*. Double-clicking a particular *Eyelimage* will zoom that *Eyelimage* and also select (and highlight) that *CalibrationDataPoint* in the *EyeSpace* window. Double clicking the zoomed image will go back to the matrix view. Alternately, selecting a *CalibrationStimulusPoint* in the *EyeSpace* window will zoom to that *Eyelimage* in the *CalibrationImage* window. If an *Eyelimage* is already zoomed, then moving the calibration "DataPoint" slider will show the correct zoomed *Eyelimage*.

The CalibrationImage matrix is saved in a picture file in the ViewPoint/Calibration/ folder. In binocular mode a separate picture matrix is saved for each eye, e.g.:

ViewPoint/Calibration/vpx_EyeACalibrationImage_LastRun.bmp
 ViewPoint/Calibration/vpx_EyeBCalibrationImage_LastRun.bmp



The Calibration Eye Images in these files are written over each time a new calibration is performed including a Re-Present. The user may however make a copy of the file to save a particular set.

8.6 Binocular Calibration

In binocular mode a pull-down menu at the bottom of the *EyeSpace* window allows calibration actions may be performed on either both eyes at the same time, or on individual eyes. This is of particular use when you need to *Re-present* or *Omit* a point to only one eye.

8.7 Omitting Individual Calibration Points

8.7.1 Automatic Omitting

During automatic calibration, *ViewPoint* automatically omits calibration DataPoints that have bad data quality codes. *ViewPoint* removes as many bad data quality calibration DataPoints as possible, until the minimum calibration DataPoints required is reached. Any automatically omitted calibration DataPoints can be re-presented or re-instated at any time.

8.7.2 Manual Omitting

The *EyeSpace window > Omit button* in the will allow the user to omit an individual calibration DataPoint from the mapping calculations. This may be required if for some reason an individual calibration point is far out of the rectilinear distribution and re-present is not successful. Use the DataPoint slider or mouse click to select the required DataPoint. Pressing the *EyeSpace window > Reinstate button* reinstates the omitted point. When in binocular mode, calibration points can be omitted for either or both eyes.

8.8 Re-presenting Individual Calibration DataPoints

The DataPoint slider allows the user to select individual (e.g., stray) calibration points to be recalibrated. The active DataPoint is highlighted in the graphics well. DataPoints can also be selected by left clicking the mouse. To re-present the selected (stray) calibration point, press the *EyeSpace window > Re-present button*.

The message “Get Ready” will appear briefly on the screen to draw the subject’s attention to the location of the calibration point. This can be suppressed or the display time lengthened via the Advanced section in the *EyeSpace* window.

8.9 Slip Correction

During data collection, the subject may move in the X and Y planes such that the measured Position of Gaze no longer corresponds to actual Position of Gaze. This type of problem can usually be corrected easily by translating (shifting) the calibration data set. First be sure to select (click with the mouse) a good calibration point near the center of the display. The *EyeSpace window > Slip-Correction button* will represent the currently selected calibration point to the subject and automatically adjust the remaining points to compensate for the measured slip in the (x,y) plane.

The message “Get Ready” will appear briefly on the screen to draw the subject’s attention to the location of the calibration point. This can be suppressed or the display time lengthened via the Advanced section in the *EyeSpace* window or by using *Settings* files. Refer to section [16.3](#).

Slip-Correction is generally not required when using the *Pupil-Glint Vector Difference* method; it is most useful when using the pupil-only or glint-only methods.



Slip-Correction is a different from *Slip-Compensation* feature method described in section [6.9](#).

8.10 Gaze Nudge

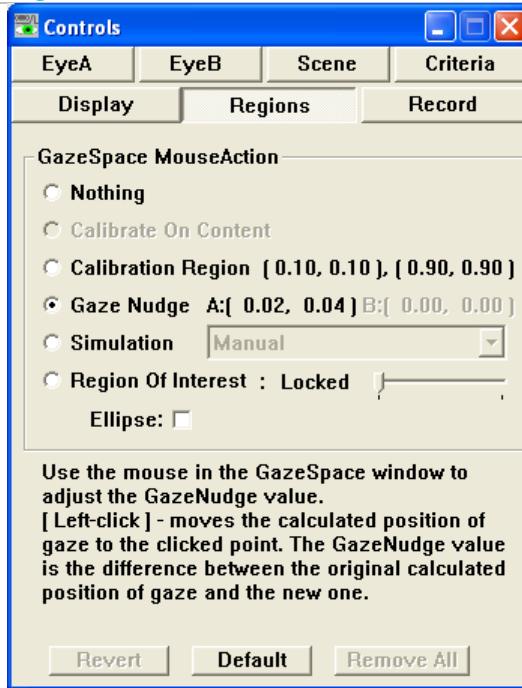
Gaze Nudge adds an (x,y) nudge vector to the calculated position of gaze (POG) so it appears at the correct location. In non-averaged binocular mode separate nudge vectors are applied to the POG of each eye. *Gaze Nudge* is very similar to *Slip Correction* except that the correction for *Slip Correction* is done in the *EyeSpace*, which changes the mapping function, and the correction for *Gaze Nudge* is done in the *GazeSpace* by simply applying the *Gaze Nudge* offset.

Gaze Nudge appears in all the *Corrected* data throughout *ViewPoint*. This includes, but is not limited to, the *PenPlots*, the *DataFile*, and the DLL access functions. Also, when the *Gaze Nudge* is modified during a recording, its new values are interleaved between data records,

e.g. `eyeA:gazeNudge 0.35 0.16.`

ROI calculations are performed after nudging vectors and averaging is applied, so what you see is what you get.

Figure 16. Gaze Nudge



There are two CLI commands: `GazeNudge xPos yPos` and `GazeNudgeInc xInc yInc`, see [19.12.29](#) for details.

The nudge is applied only to the *Corrected* data; see section [14.2.1, Unprocessed & Corrected Data](#).

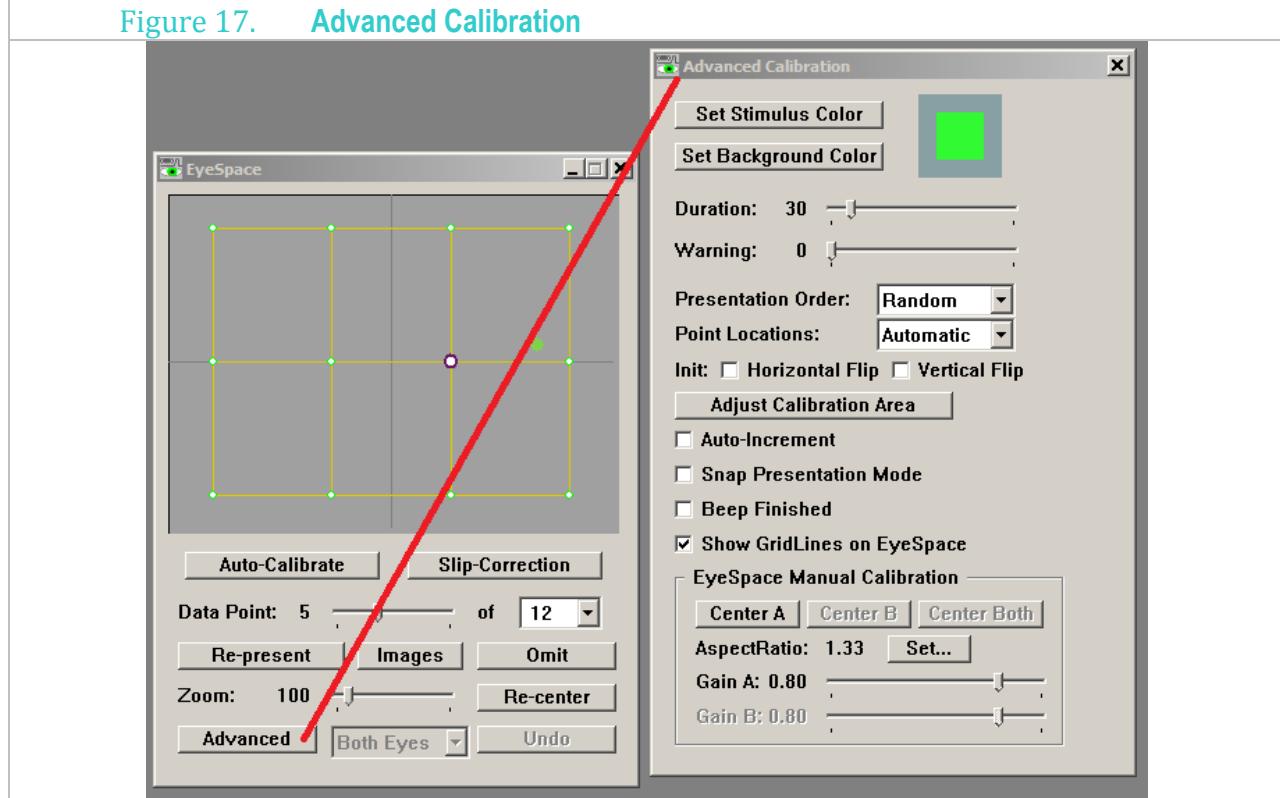
8.11 Dominant Eye

With a monocular eye tracker, if the subject is known to have a dominant eye, a better calibration is obtained if this dominant eye is used.

8.12 Advanced Calibration Controls

The *Advanced Calibration* window allows the user to customize many calibration features along with fine tuning the manual calibration settings. You can launch the *Advanced Calibration* window, shown in [Figure 17](#), by clicking the *EyeSpace window > Advanced button*.

Figure 17. Advanced Calibration



8.12.1 Calibration Stimulus & Background Color

The user can change the default color settings of the calibration stimulus rectangles and the background using the [*Set Stimulus Color*] and the [*Set Background Color*] buttons.

8.12.2 Timing & Warning

The message “**Get Ready**” will appear briefly on the screen to draw the subject’s attention to the location of the calibration point. This can be suppressed or the display time lengthened via the *Warning slider* or CLI: `calibration_WarningTime <value>`.

The *Duration slider* or CLI: `calibration_StimulusDiration <value>` specifies the approximate duration in milliseconds of each of the concentric contracting “tunnel motion” calibration stimulus rectangles. CLI: `calibration_ISI <value>` specifies the inter-stimulus interval between calibration points.

8.12.3 Presentation Order

The *Presentation Order* pull-down menu and CLI `calibration_PresentationOrder` allow the user to present the calibration StimulusPoints in one of three ordering modes. The default presentation mode is *Random*.

- ④ *Sequential*: Calibration stimulus rectangles are presented from the top left hand corner of the screen to the bottom right hand corner of the screen, one column at a time.
- ④ *Random*: Calibration stimulus rectangles are presented in random order. The series is re-randomized every time the set finishes, so that there is a new random order for the next loop.



- ④ *Custom*: Using CLI commands with `calibration_CustomOrderList`, the user can specify the presentation order of the calibration stimulus rectangles; see sections [19.12.14](#).

8.12.4 Stimulus-Point Locations

Normally for a HeadFixed system the locations of the calibration stimulus points are automatically positioned such that they are evenly spaced within the calibration region. Sometimes because of occluders (things that get in the way) or other special situations these need to be moved. Sometimes they need to be positioned based upon the content. The *Point Locations* pull-down menu and CLI `calibration_PointLocationMethod` allow users to specify how the locations of the Calibration Stimulus Points will be determined:

- ④ *Automatic* : HeadFixed – creates an evenly spaced grid within the Calibration Region.
- ④ *Custom* : HeadFixed – the user specifies the locations of the calibration StimulusPoints using CLI command `calibration_CustomPoint` for each point; these are typically listed in a *Settings* file. See example in section [8.12.4.2](#).
- ④ *OnContent* : This allows the user to specify the position of custom calibration StimulusPoints at locations in the *SceneCamera (GazeSpace)* window based on the video content (e.g. a finger, a cardboard plaque with calibration dots painted on it etc.) When in this mode, a right mouse click in the *GazeSpace* window does two things:
 - (a) Sets *Custom* Calibration *StimulusPoint* at the point the user clicks in the *GazeSpace* window, on scene video content. Saving a *Settings* file will save these interactively specified Custom points, so they can be loaded again.
 - (b) Sets calibration *DataPoint* immediately without showing a calibration *Stimulus Point*, i.e. in snap fashion.

Side effects of these changes are listed in section [19.12.19](#).

8.12.4.1 Custom Calibration Point Positions

ViewPoint allows the user to specify the locations of the calibration StimulusPoints. This can be very useful if you need to avoid certain visual obstacles (occluders).



You must set `calibration_PointLocationMethod` `Custom` before loading custom calibration Stimulus Points; otherwise the locations will not persist.

The nearest-neighbor grid-lines in the *EyeSpace* are only useful if the pattern of StimulusPoints is in a rectilinear grid. The nearest-neighbor grid-lines are not automatically drawn when this option is used, because the points could be in any configuration. The drawing of these lines can be toggled on/off with CLI `calibration_ShowEyeSpaceGrid <bool>`. See section [19.12.22](#).

The calibration point index is in column-major order, which means that the points run along the columns before going to the next row.

8.12.4.2 Partial Binocular Overlap

Calibration can be performed on Partial Binocular Overlap (PBO) Head Mounted Display (HMD) systems by specifying different sets of custom calibration points for each of the two eyes. See section [5.5.1.1](#) for more discussion and section [19.12.20](#) for more detailed example code. See also section [12.5](#) about Stereo Display of stimuli. Note the EyeA: and EyeB: prefix in the following example:

```
stereoDisplay ON
calibration_PointLocationMethod Custom
EyeA:calibration_CustomPoint 1 0.1 0.10
EyeB:calibration_CustomPoint 1 0.1 0.15
etc.
```

8.12.5 Flipping the Initial Calibration

Check the `Init: [x] Horizontal Flip` and the `[x] Vertical Flip` check boxes as appropriate when using mirrors or with rotated cameras so the eye movement directions are correct in the initial calibration mapping. These specify whether the initial calibration should be created as mirror reflections of the defaults. Setting these correctly is especially important when using *Manual Calibration*. These flip options persist during Manual Calibration adjustments.

Below are the CLI commands for these features.

```
calibration_InitWithHorizontalFlip <bool>
calibration_InitWithVerticalFlip <bool>
```

Table 6. Init with Horizontal & Vertical Flip options

<i>No Flip</i>	<i>Horizontal Flip</i>	<i>Vertical Flip</i>	<i>Horizontal & Vertical Flip</i>
10 7 4 1	1 4 7 10	12 9 6 3	3 6 9 12
11 8 5 2	2 5 8 11	11 8 5 2	2 5 8 11
12 9 6 3	3 6 9 12	10 7 4 1	1 4 7 10



The calibration flipping is different from the option of mirroring the *EyeCamera* image, which is available on some products, see CLI: `videoMirror`.

8.12.6 Adjusting the Calibration Area

The user may want to adjust the size and position of the area within which the calibration StimulusPoints are presented. This is especially useful when part of the display screen is occluded, as in fMRI environments. This opens the *Controls window > Regions tab*. Use the left mouse button in the *GazeSpace* window to drag out the required size and position of the calibration area. The size and position coordinates are displayed in the *Regions tab* and in the *GazeSpace* window. The *Revert* button will undo the last change and the *Default* button will return the calibration areas size to the default setting.

8.12.7 Snap and Increment Calibration Modes

If you prefer to calibrate points individually, then the user can select the *[x] Snap Presentation* mode checkbox. In this mode the currently selected calibration DataPoint is active and the *Re-present* button will immediately perform the calibration based on the eye position at the moment the button is pushed. There is no warning and no calibration StimulusPoint presented.

When in this mode the behavior of the *Re-Present* and the *Slip-Correction* buttons are changed as described and is indicated by the appearance of an asterisk (*) on these buttons.

When operating in this mode the user can choose whether to automatically advance to the next calibration DataPoint by selecting the *Auto-Increment* button. This mode is indicated by the appearance of a double plus icon (++) on the *Re-Present* and the *Slip-Correction* buttons. If *Auto-Increment* is not selected then the selected calibration point will not change. The successor number DataPoint is determined by the *Presentation Order* mode selected.

8.12.8 Manual Calibration

The *Advanced Calibration* window now includes GUI controls for manual calibration. Manual calibration is particularly useful for non-verbal subjects. The buttons and sliders for doing this are in the *Advanced Calibration* window, at the bottom in the *EyeSpace Manual Calibration* group box; See [Figure 17](#).

This is done in three steps.

- ④ Set the *AspectRatio* of the *Stimulus* window, if it is different than the default aspect ratio of the *EyeSpace* (4:3 = 1.33), by pressing the *Aspect Ratio: n.n Set...* button.
- ④ Set the *Center* point when you believe the subject is looking at the center of the screen, by pressing the *Center A* (*Center B*, or *Center Both* for binocular systems) button.
- ④ The user can change the amount of *EyeSpace* movement that corresponds to *GazeSpace* movement by adjusting the *Gain A* (and *Gain B* for binocular systems) slider.

Adjusting the gain value modifies the distribution of the *EyeSpace* calibration data points, which determines the mapping from *EyeSpace* to *GazeSpace*. The calibration data points can be saved and reloaded as usual, thus saving and reloading the manual calibration.

In addition, the *Init: [x] Horizontal Flip [x] Vertical Flip* check boxes allow the user to specify whether the initial calibration should be created as mirror reflections of the defaults. This only effects the initial ideal calibration data; it does not effect subsequent calibration on a subject.

Below are the available CLI commands for these features.



```
calibration_InitWithHorizontalFlip <bool>
calibration_InitWithVerticalFlip <bool>
manualCalibration_AspectRatio <float>
eyeA:manualcalibration_Center
eyeB:manualcalibration_Center
both:manualcalibration_Center
eyeA:manualCalibration_Gain <float>
eyeB:manualCalibration_Gain <float>
both:manualCalibration_Gain <float>
```

8.12.9 Calibration Points in a *Settings* file

Both calibration StimulusPoints and calibration DataPoints for each eye are saved to *Settings* files.

The columns are:

fm: the *FeatureMethod* 0=PupilOnly, 10:GlintOnly, 20=PupilGlintVectorDifference.

ix: the calibration point number from 0 to (n-1).

XeyeDat and **YeyeDat**: the calibration *DataPoints*.

XstimPt and **YstimPt**: the calibration *StimulusPoints*.

The following is taken from a *Settings* file:

```

EyeA:// CALIBRATION DATA (UNIFIED) -----
EyeA:// fm=0:pupilOnly, fm=10:GlintOnly, fm=20:PupilGlintVectorDifference
EyeA://----- fm ix XeyeDat YeyeDat XstimPt YstimPt Omit
EyeA:CalibData 0 0 0.90000 0.10000 0.10000 0.10000 0
EyeA:CalibData 0 1 0.90000 0.50000 0.10000 0.50000 0
EyeA:CalibData 0 2 0.90000 0.90000 0.10000 0.90000 0
EyeA:CalibData 0 3 0.50000 0.10000 0.50000 0.10000 0
EyeA:CalibData 0 4 0.50000 0.50000 0.50000 0.50000 0
EyeA:CalibData 0 5 0.50000 0.90000 0.50000 0.90000 0
EyeA:CalibData 0 6 0.10000 0.10000 0.90000 0.10000 0
EyeA:CalibData 0 7 0.10000 0.50000 0.90000 0.50000 0
EyeA:CalibData 0 8 0.10000 0.90000 0.90000 0.90000 0
EyeA:CalibData 10 0 0.90000 0.10000 0.10000 0.10000 0
EyeA:CalibData 10 1 0.90000 0.50000 0.10000 0.50000 0
EyeA:CalibData 10 2 0.90000 0.90000 0.10000 0.90000 0
EyeA:CalibData 10 3 0.50000 0.10000 0.50000 0.10000 0
EyeA:CalibData 10 4 0.50000 0.50000 0.50000 0.50000 0
EyeA:CalibData 10 5 0.50000 0.90000 0.50000 0.90000 0
EyeA:CalibData 10 6 0.10000 0.10000 0.90000 0.10000 0
EyeA:CalibData 10 7 0.10000 0.50000 0.90000 0.50000 0
EyeA:CalibData 10 8 0.10000 0.90000 0.90000 0.90000 0
EyeA:CalibData 20 0 0.90000 0.10000 0.10000 0.10000 0
EyeA:CalibData 20 1 0.90000 0.50000 0.10000 0.50000 0
EyeA:CalibData 20 2 0.90000 0.90000 0.10000 0.90000 0
EyeA:CalibData 20 3 0.50000 0.10000 0.50000 0.10000 0
EyeA:CalibData 20 4 0.50000 0.50000 0.50000 0.50000 0
EyeA:CalibData 20 5 0.50000 0.90000 0.50000 0.90000 0
EyeA:CalibData 20 6 0.10000 0.10000 0.90000 0.10000 0
EyeA:CalibData 20 7 0.10000 0.50000 0.90000 0.50000 0
EyeA:CalibData 20 8 0.10000 0.90000 0.90000 0.90000 0
// -----

```

The locations are in normalized coordinates, so for example in a window 800 x 600, the (x,y) locations (0.1,0.1) are at ((0.1*800), (0.1*600)) that is (80,60).

Chapter 9. Corrections Based on Measurements

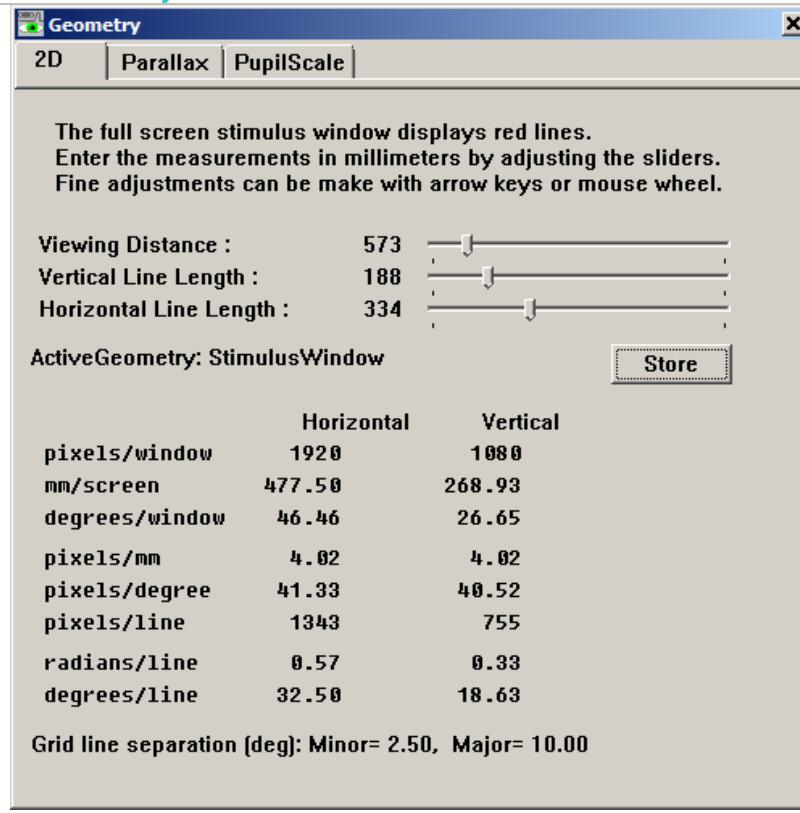
The previous chapter described mapping to the calculated *Position of Gaze* (POG). This chapter describes additional corrections that can be applied based on geometric measurements.

9.1 Obtaining POG in Degrees: 2D Geometry

By default, the calculated *GazePoint* is given in normalized window coordinates (that is: (0.0,0.0) at the top left, (0.5,0.5) in the center, and (1.0,1.0) at the bottom right). These values are always correct, regardless of the display resolution settings, and regardless of whether or not the user has correctly measured and entered the geometry distances.

To obtain the *GazePoint* in degrees, you will need to make three measurements and enter these measured values by adjusting the three sliders in the *Geometry window > 2D tab*. One measure will be of the viewing distance that is the distance from the eye to the display; the other two measures will be of horizontal and vertical *MeasurementLines*. Don't worry, *ViewPoint* does the trigonometry for you! The 2D Geometry is now modal, that is, the values that appear depend upon which mode you are in. The mode is displayed as the *Geometry window > 2D tab > ActiveGeometry: value*. The *ActiveGeometry* (mode) is set according to which *Stimuli > View Source* is selected (*ViewPoint Stimulus Window*, *Head Mounted SceneCamera*, or *Interactive Computer Display*) and which display device is selected (*Monitor 1*, *Monitor 2*, etc.) or if the *Stimulus* window is not full screen.

Figure 18. Geometry Window



How and where you make the measurements depends on what type of system you are using, as explained in the following sections:

9.1.1 Stimulus Window or InterActive Display -- HeadFixed (non-HMD)

First show the Stimulus window as it will be used (e.g. full screen on a single monitor), then raise the *Geometry window > 2D tab* so it appears over the Stimulus window. In this mode, *MeasurementLines* are drawn in the Stimulus window whenever the *2D panel* is active.

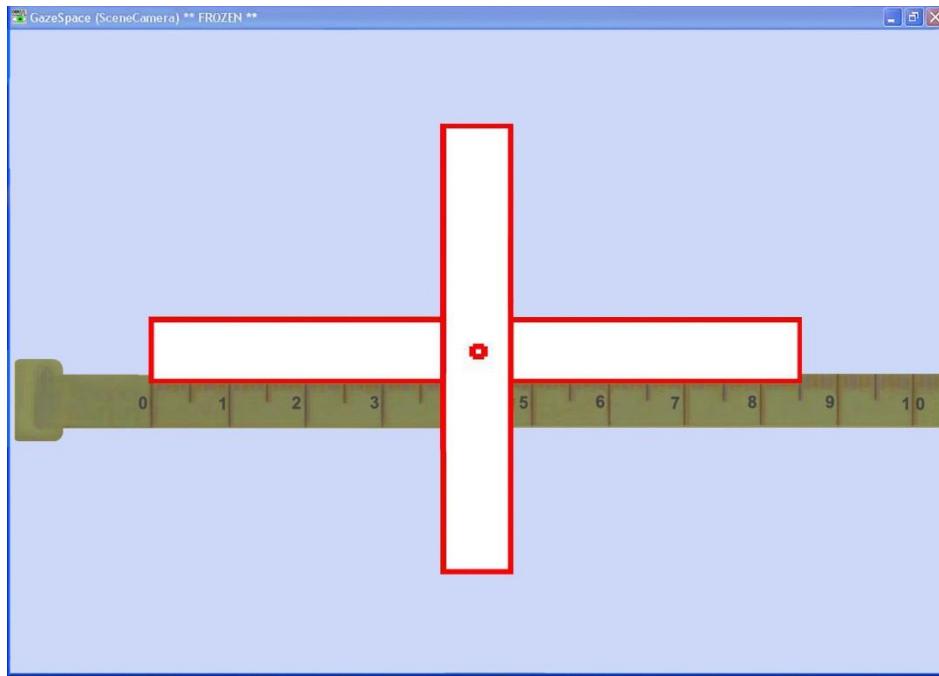
Using a tape measure, determine the length of each of the *MeasurementLines* including the red border and also the distance from the eye to the center of the Stimulus window, at the intersection point of the *MeasurementLines*. Enter these measured values by adjusting the three sliders in the *Geometry window > 2D tab*.

9.1.2 HeadMounted SceneCamera

When using the *SceneCamera* you will be looking at the video in the *GazeSpace* window (rather than the *Stimulus* window). While the video is being displayed, raise the *Geometry window > 2D tab*. In this mode, *MeasurementLines* are drawn in the *GazeSpace* window whenever the *2D panel* is active. It is best to make

the measures relative to a wall or a vertical board. This can be done with a single person using a comfortable distance (within arm's reach) (see section [9.1.4](#), below, regarding distances). Using a tape measure, first measure the distance from the eye (*SceneCamera*) to the wall, then keeping this distance constant, extend the tape measure against the wall so that it extends along the length of the *MeasurementLines*, as viewed in the *GazeSpace* window -- enter these measured values by adjusting the three sliders in the *Geometry window > 2D tab*. Refer to [19.4](#)

Figure 19. Read the Measuring Tape in GazeSpace/SceneCamera content



9.1.3 HeadMounted Display

Because taking useful measurements inside the HMD is extremely difficult, we will instead take the measurements from after-images created by the *MeasurementLines* shown in *Stimulus* window in the HMD. You will need to get the following things ready, before you begin:

- (a) One clean white sheet of paper with a fixation dot in the middle.
- (b) A pencil
- (c) A short ruler (e.g. 7")



Read through and understand these instructions before starting the measurement procedure

Mark the center of the sheet of paper with a dark dot suitable for fixation (e.g. about 2.5 mm). Hold the ruler upright (vertically) against the paper, such that the forehead can rest against the top of the ruler, keeping the head a constant distance from the paper.

Now, show the *Stimulus* window in the HMD, then raise the *Geometry window > 2D tab*, which will display *MeasurementLines* in the *Stimulus* window shown in the HMD.

Stare exactly at the center i.e., at the intersection of these lines for at least about 15 seconds, without moving the eyes.

As quickly as possible, (a) remove the HMD, (b) position the head at the known distance from the paper using the ruler, (c) stare exactly at the fixation spot on the paper and mark the ends of the after-image lines with the pencil.

After marking the ends, using a ruler to measure the lengths of the after-image lines. Enter these measured values by adjusting the three sliders in the *Geometry window > 2D tab*.

9.1.4 Notes on Measurement Lines

For the *Stimulus* window ([9.1.1](#)) and the *SceneCamera* ([9.1.2](#)) we do not directly measure the display hardware, because the display driver could have been set to shift part of the image off the screen, or it could have been shrunk or expanded to better fit a particular monitor. Rather, we measure the vertical and horizontal *MeasurementLines* that cover about 80% of the window.

For the *SceneCamera* ([9.1.2](#)) and HMD ([9.1.3](#)) the viewing distance does not matter, because the lengths of the *MeasurementLines* vary directly with changes in the viewing distance -- the angles will be the same.



The units of measurement are arbitrary, but they must be consistent, e.g. all measurements in millimeters, all measurements in inches, etc...

The *Geometry* window also displays various numerical calculations that may be useful.

After adjustments have been entered, the measurements should be saved by pressing the *Store* button. Subsequent runs of *ViewPoint* will maintain the stored settings.

The *Active Geometry* text shows which set of geometry values are being used. Settings are stored separately for different modes of operation, for example when the *View Source* is changed between *SceneCamera* and *Stimulus* window. Changing modes automatically loads in the stored values for that mode.

9.1.5 Geometry Grid

After the geometry has been measured and entered (see section [9.1.1](#)), the *GeometryGrid* lines will be accurately spaced to show the viewing angle in degrees of visual arc.

The *GeometryGrid* can help the user understand the sizes of and the distances between visual stimuli presented in the *Stimulus* window; it can also be useful for assessing the accuracy of the calibration that was obtained

This *GeometryGrid* display shows light blue vertical and horizontal lines and circles radiating from the center, separated by a specifiable number of degrees of visual arc. The user can specify the spacing of two sets of lines, minor (thin) lines and major (thicker) lines.

These values can also be set using CLI commands. See section [19.4.3](#).

The geometry gridlines can be displayed on the *Stimulus* window and on the *GazeSpace* window without the *Geometry Window* being active, by using the check boxes on the *Controls window > Display tab*. See also [19.4.3](#).

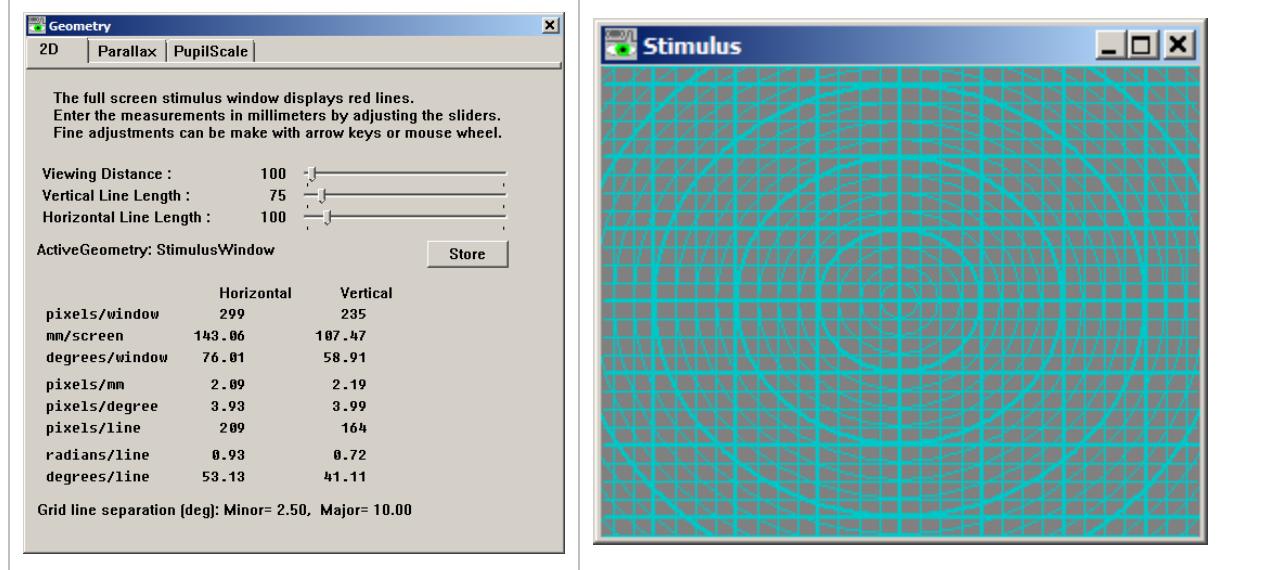


Arrington Research



For *Stimulus* window (9.1.1) and HMD (9.1.3) modes, the grid lines are accurate only on the *Stimulus* window and only at the specified viewing distance; the grid lines shown in the *GazeSpace* window are scaled and are typically a miniature view of the *Stimulus* window.

Figure 20. Geometry Window



You can specify the minor and major axis separation displayed using the CLI: `gridSpacing floatMinor floatMajor`

e.g.: `gridSpacing 2.5 10 // minor axes every 2.5 degrees, major axes every 10 degrees`

9.2 Parallax Correction for Binocular SceneCamera systems



This section is only applicable to *Binocular SceneCamera* systems.

When the optical axis of the *SceneCamera* is different from the optical axis of the *Eyeball*, a parallax error will occur in the calculated Position of Gaze (POG) when the subject is looking at points in a plane that is closer or farther away from the plane of calibration. The binocular *SceneCamera* system provides a built-in default parallax correction and also the ability to fine tune the correction.

The parallax correction value is a linear function of the binocular vergence angle. The slope of the line depends on the inter-pupillary distance (IPD) of the eyes and the distance of the *SceneCamera* above or below the line between the two eyes. In general, the default value offers substantial correction value, based on the typical camera location on the *ViewPoint EyeTracker* ® *EyeFrame* ™ and typical IPDs .

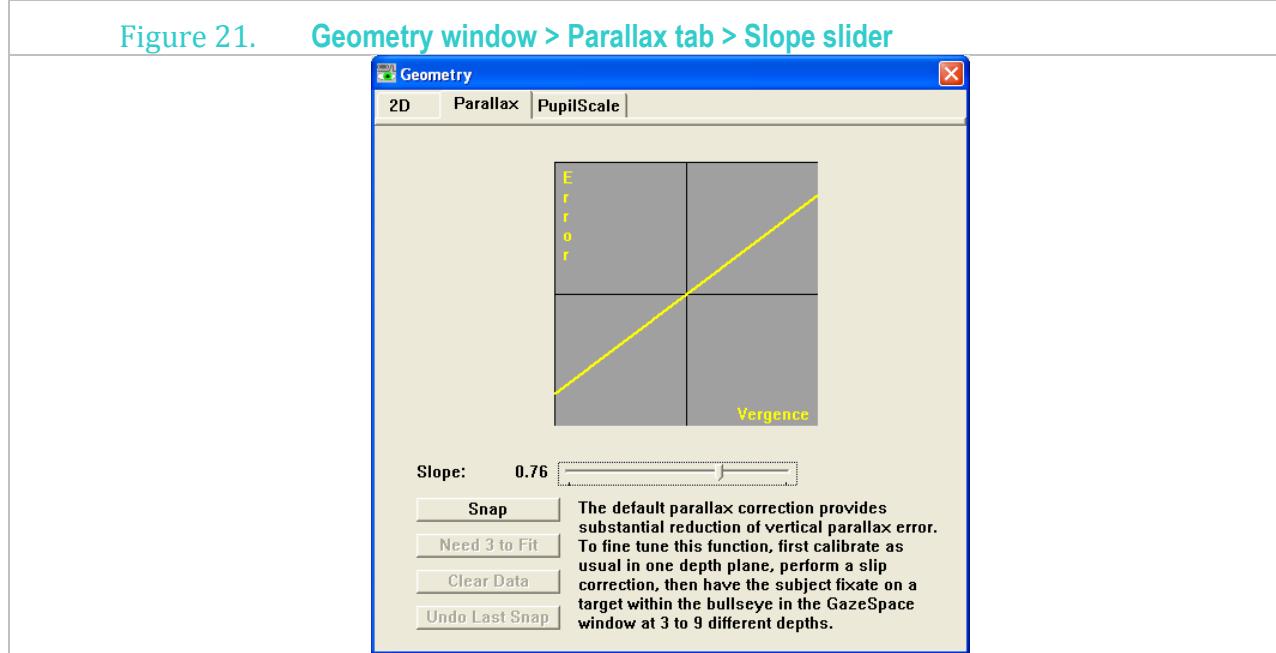
To show the Position of Gaze with the Parallax correction, select menu item: *Binocular > Show average with parallax correction*. Parallax Correction will not be applied to the data unless both *Binocular mode* and *SceneCamera mode* is active, and *Parallax Correction* is selected.

If you wish to fine tune this function, there are two ways to determine the appropriate parallax correction: (a) manual experimentation, (b) collecting data and fitting a line to it.



CRITICAL: Always do an *EyeSpace Slip-Correction* before calibrating the parallax correction.

After performing the *Binocular SceneCamera* calibration, bring up the *Geometry window > Parallax tab* as in [Figure 21](#) below.



9.2.1 Manual Parallax Adjustment

Use the Slope slider to adjust the magnitude of the Parallax Correction function, as the subject looks near and far, until a setting is found that substantially eliminates the error between the supposed gaze point and the calculated gaze point.

The default slope applied is 0.82, if you need to change this default use the following CLI command:

```
parallaxCorrection_Slope float // where float is in range -2.0 – + 2.0
```

9.2.2 Parallax Correction from Data

Here's how to perform the calibration:

1. Place the target in the bulls eye located in the *Gaze Space window* as see in [Figure 22](#).
2. Ask the subject to fixate on the actual target. For example, in the image in [Figure 22](#) below, if the subject was ask to look at the tip of the finger.
3. Once the subject is fixating on the target, press the [*Snap*] button. You should see a new red data point appear in the plot. The software records the difference between the *calculated Position of Gaze* (POG) and the *target* to get the error term, the current vergence angle is also recorded.
4. Repeat steps 1 - 3 for a minimum of 3 different distances, E.g. 10cm, 20cm, 1m, 2m, 3m. A minimum of 3 points are required before you will be allowed to fit a line to the data; in fact you should usually obtain at least 6 - 9 for a more accurate calibration.

If at anytime you adjust the slope using the slider, you can revert back to the collected data points by pressing the [*Use Data Fit*] button.

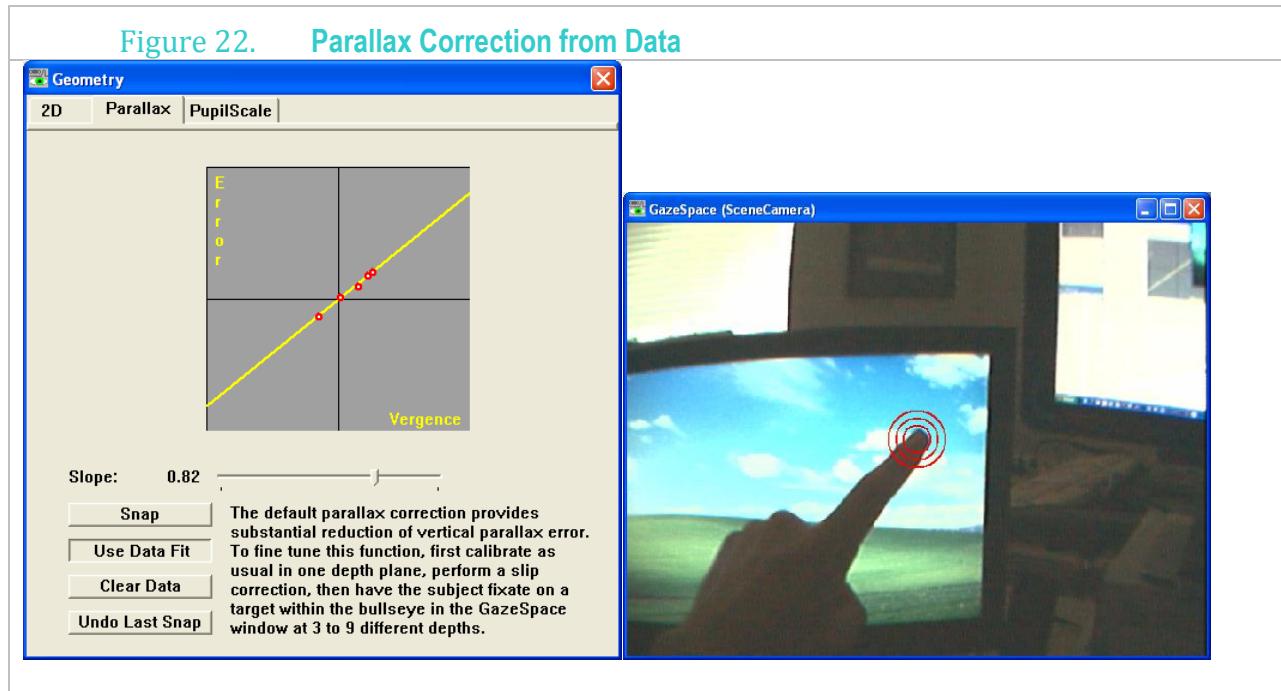
```
parallaxCorrection_Snap      ==  [ Snap ] button
parallaxCorrection_UseDataFit ==  [ Use Data Fit ] button
parallaxCorrection_InitData   ==  [ Clear Data ] button
parallaxCorrection_UndoLastSnap ==  [ Undo Last Snap ] button
```

In previous versions the user was required to mouse click on the target location in the *GazeSpace / SceneCamera* window.

The parameter *Parallax* is no longer valid for *GazeSpace_MouseAction*, instead use

GazeSpace_MouseAction Simulation; simulationMode Pattern; use *GazeSpace_MouseAction None*;

To facilitate transition; this command will now do the same as CLI *geometryTab Parallax*.



9.3 Pupil Diameter

9.3.1 Raw Pupil Size

The value of *PupilWidth* is width of the ellipse that is fit to the pupil, normalized with respect to the width of the *EyeCamera* window. Using normalized values allows the user to switch between video modes (e.g. 320x240, or 640x480) without affecting the data. Consequently, if the pupil ellipse horizontally spans the full width of the window, the width value would be 1.0.



Arrington Research

The Ellipse method fits a rotated ellipse to the pupil; with this method the PupilWidth is the length of the major axis (the longest axis). Note that the minor axis will become small as the eyeball rotates away from the optical axis of the camera, as a cosine function of rotation, becoming zero at 90 degrees. The major axis does not suffer from this problem. Note also that the rotated ellipse allows a diagonal fit to the full size of the *EyeCamera* window, which allows the PupilWidth value to exceed 1.0.

The older OvalFit method fits an unrotated ellipse to the pupil; with this method the PupilWidth is the horizontal width of the pupil.

Obviously the actual pupil diameter will depend on the camera placement relative to the eye. [Figure 23](#) Artificial Pupil Diameters contains a set of black disks of specific diameters that can be used as “artificial pupils” for determining what actual pupil diameter (in inches or millimeters) the diameter in pixels corresponds to. Pupil diameters usually range between 2mm and 8mm.

Figure 23. Artificial Pupil Diameters						
● 1/8"	● 3/16"	● 1/4"	● 5/16"	● 3/8"		
● 2 mm	● 3 mm	● 4 mm	● 5 mm	● 6 mm	● 7 mm	● 8 mm

9.3.2 Calculated Pupil Diameter

Without pupil diameter calibration, the values provided by *ViewPoint* are normalized with respect to the horizontal size of the *EyeCamera* window. To obtain actual values in millimeters, the user must calibrate; the calibration provides a scale factor (for each eye) from the normalized coordinates to the diameter in millimeters.

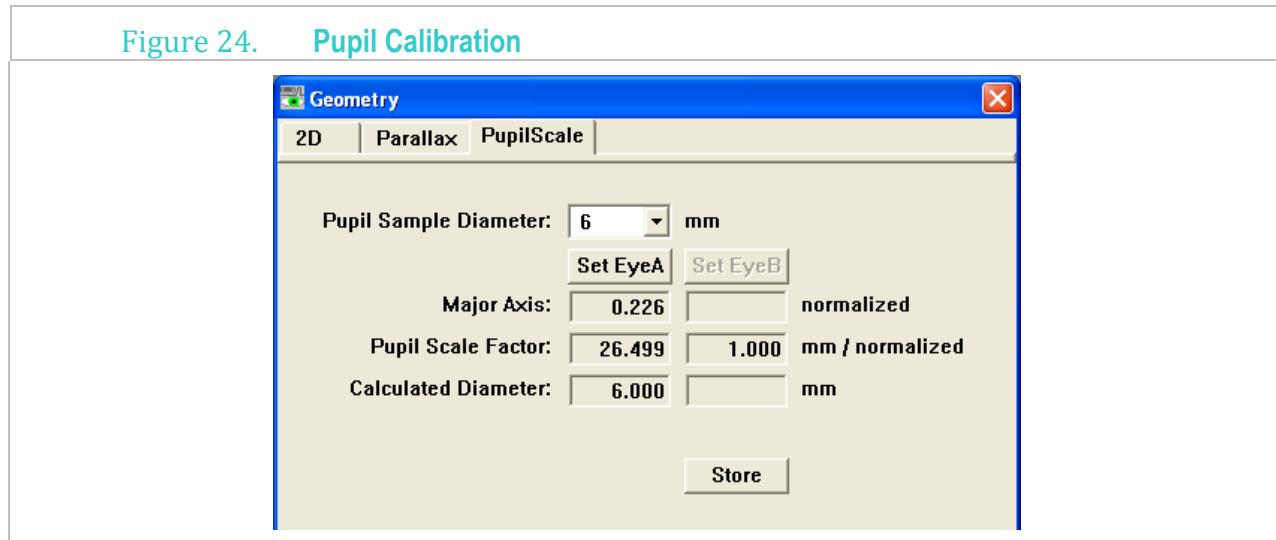
Advantages

- Easy GUI calibration
- Real-time diameter in millimeters
- DataFile contains calibrated pupil diameter in mm as well as raw data
- Calculations always uses major axis of ellipse

Procedure

1. Print out an **Artificial Pupil** template, as found in [Figure 23](#).
2. Measure it to verify that it was printed at the expected size.
3. In *ViewPoint* open the **Geometry window** by pressing “Ctrl + G”.
4. Choose an **Artificial Pupil** size and select the corresponding **Pupil Sample Diameter** in mm, from the pull-down menu combo box.
5. Place the **Artificial Pupil** template in front of the *EyeCamera* at the distance of the eye’s pupil.
6. Verify that *ViewPoint* has obtained a good elliptical fit to the **Artificial Pupil**.
7. Press the [*Set EyeA*], or [*Set EyeB*], buttons, as appropriate.
8. Press the [*Store*] button to save the results in the *Preferences* file.

Figure 24. Pupil Calibration



This pupil diameter data in mm is available in real-time in the *PenPlot window > Pupil Diameter (mm)* and via the SDK command: `VPX_GetPupilDiameter2(EYE_A, &mmPD);`
ViewPoint will insert the *Pupil Scale Factor* value into the top material of the *DataFile* and record the pupil diameter in columns APD/BPD if the user has calibrated and stored the pupil diameter. It is stored in *Settings* files as CLI: `EyeA:pupilScaleFactor floatValue`.

9.4 Inter-Pupillary Distance (IPD)

CLI: `ipdMeasure millimeters`

Accurate calculation of **3D GazePoint** and **3D Vergence** calculations require accurate `ipdMeasure`, `geoHorizontalMeasure`, `geoVerticalMeasure` and `geoViewingDistance` measurements be set.

Note that *ViewPoint* assumes that the vertical eyeball position is at the vertical center of the *Stimulus* or *SceneCamera* window. For more flexibility, use **3DViewPoint** or **3DWorkSpace** products.

9.5 Pupil Aspect

Blinks can be detected by monitoring the pupil aspect ratio. This is a dimensionless value, where 1.0 indicates a perfect circle.

Chapter 10. Cursor Control - EyeMouse

The *CursorControl* feature allows the subject to move a cursor with their eyes. This is sometimes called an *EyeMouse*. When this feature is active, the cursor is automatically moved to the calculated Position of Gaze. It only works properly with (a) head fixed hardware (e.g., the *Headlock™*) or (b) head mounted display (HMD) hardware. It does not work with the mobile *SceneCamera* systems.

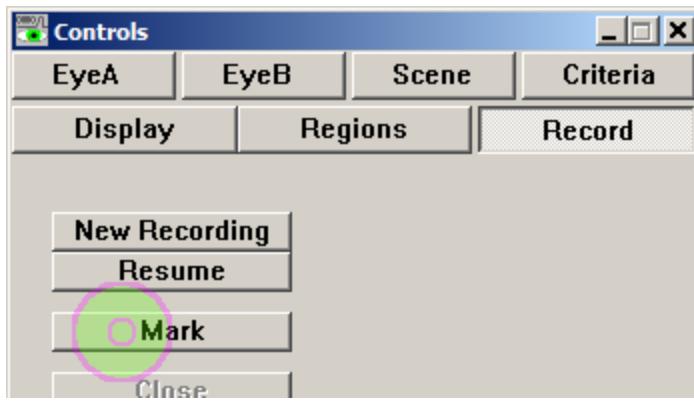
Selecting menu item: *Interface > CursorControl > GazeCursor* will cause a transparent disk (the cursor) to move over the computer displays wherever the subject is looking.

Hint: Increasing smoothing will substantially increase the usability of CursorControl.

Selecting menu item: *Interface > CursorControl > Fixation Clicks Buttons* will cause a **mouse click event** to be issued at the fixation location whenever the fixation duration exceeds a pre-set **dwell time**. The *Controls window > Criteria tab > MouseClick DwellTime slider* specifies this fixation time in seconds. The **mouse click event** can be used to press buttons, raise windows, adjust sliders, etc.

The green transparent disk (the GazeCursor) is surrounded by two red circles; one red circle stays at the perimeter of the disk, while the other red circle shrinks toward the center of the disk while the subject is fixating. The **mouse click event** is issued when the shrinking red circle reaches the center of the disk. This provides good visual feedback, so the subject can control his fixation and can avert his eyes to avoid an unwanted mouse click. See [Figure 25](#).

Figure 25. GazeCursor - EyeMouse



Selecting: *Interface > CursorControl > Eye Moves Mouse* causes the actual computer pointer to move. CLI commands are described in section [19.22](#) and summarized here below.

<i>Interface > CursorControl > GazeCursor</i>	<code>gazeCursor_Used YES; gazeCursor_transparency 127</code>
<i>Interface > CursorControl > Eye Moves Mouse</i>	<code>cursor_Control ON // the computer pointer</code>
<i>Interface > CursorControl > Fixation Clicks Buttons</i>	<code>cursor_DwellClick ON; cursor_DwellSeconds 3.5</code>
<i>Interface > CursorControl > Blinks Click Buttons</i>	<code>cursor_BlinkClick ON</code>

Chapter 11. Ocular Torsion

11.1 Introduction to Torsion

Ocular Torsion is the rotation of the eye ball about the line-of-sight, i.e. rotation about the z-axis. *ViewPoint* measures ocular torsion by determining the rotation of the iris striation patterns. A representative striation pattern *sample* is stored as the *template* (presumably taken when the eye was at zero degrees torsion). Subsequent samples are compared against the template to determine how much rotation has occurred. Note that there is no absolute position that is clearly zero degrees, however a mean center position can be approximated by looking at the range of movement in the data.

To open the *Torsion* window Select: *EyeCamera window > Monitor Icon > Torsion*. The *Torsion* window is shown in [Figure 26](#), below. The *Torsion* window contains three graphics wells and various controls.

The line graph in the top, **Correlation**, graphicsWell in the *Torsion* window is the most important to monitor. The goal is to get a single hump that touches the top; this is the autocorrelation maximum and is the calculated amount of torsion. Normally the line is green, but it turns red when a *Torsion (deg): Range Error* occurs. A range error occurs is when the calculated torsion angle is outside the specified sampling range. Autocorrelation is a computationally expensive process and there is a tradeoff between resolution and range. It should not be necessary, but the resolution and range can be modified using CLI commands, see section [19.21](#).

Sampling begins at the most counter-clockwise point on the arc (these points will be at the left on the plots in the *Torsion* window graphics wells) and then proceeds clockwise. If you hit a glint it will usually show as a spike in the middle graphicsWell that shows the Current sample.

To start torsion measurement, press the *Start* button. When torsion is being calculated, the *EyeCamera* window will contain additional overlay graphics that indicate the arc along which the iris striations are sampled. This SamplingArc (red overlay graphics) is shown in [Figure 27](#) below.

Figure 26. Torsion Window

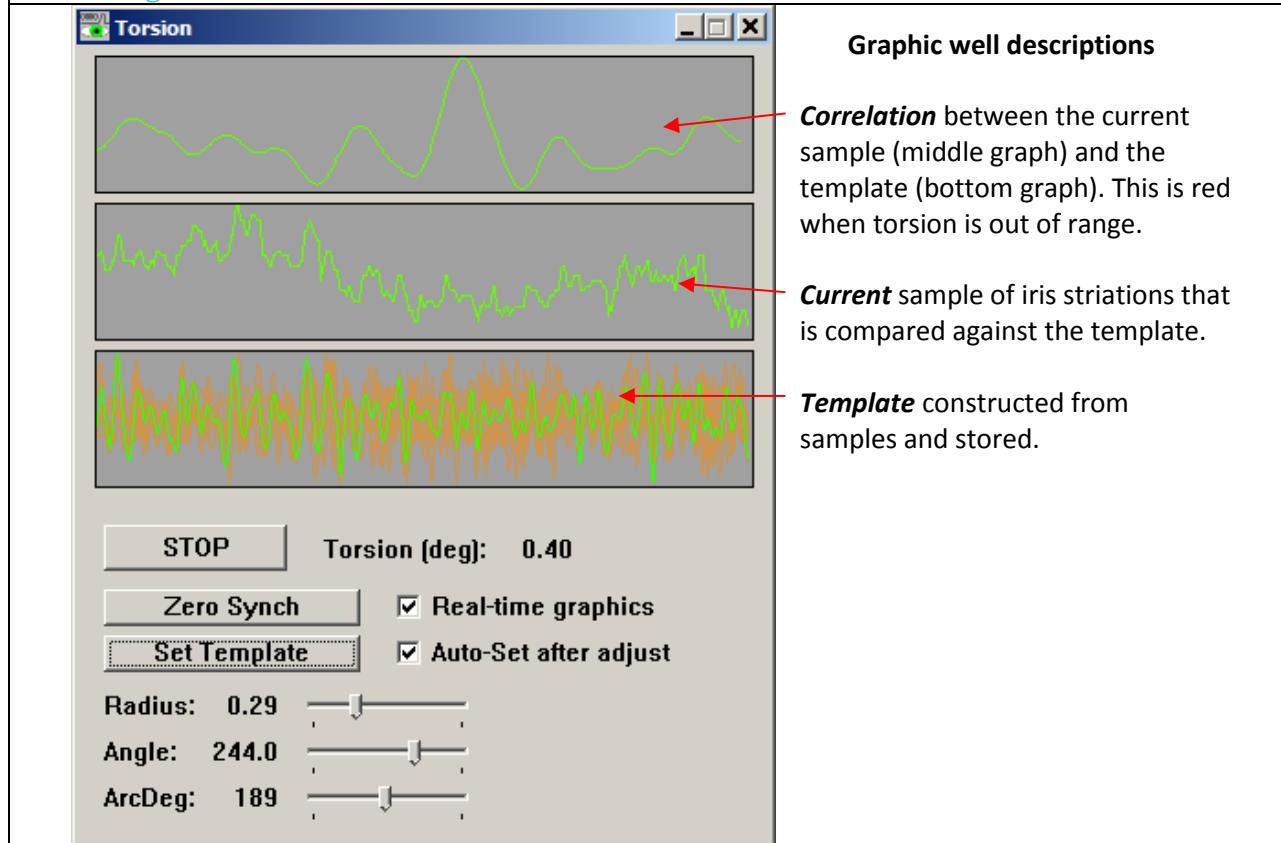
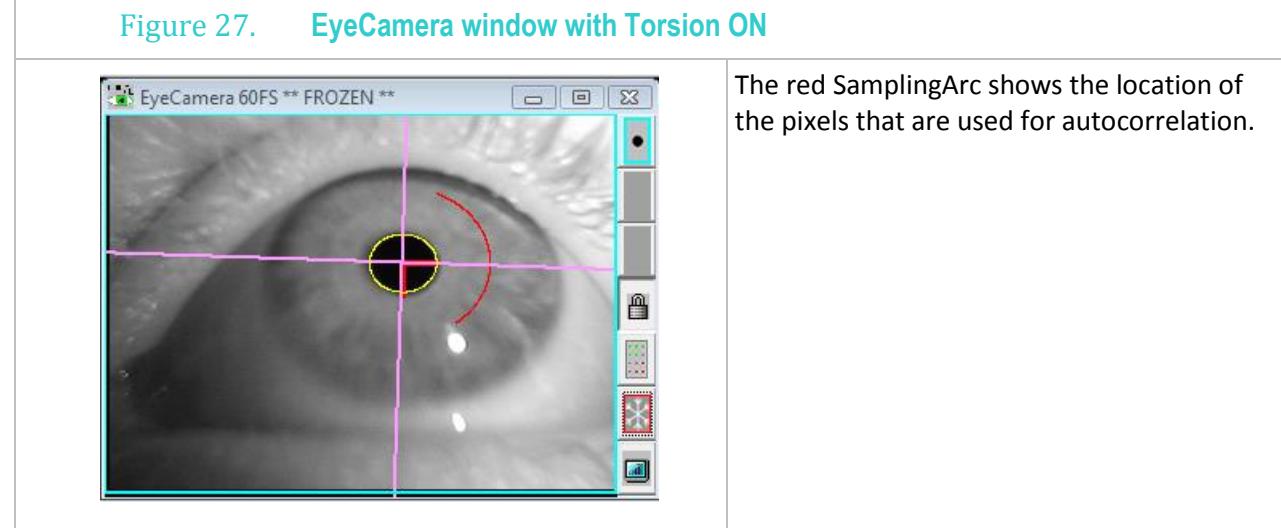


Figure 27. EyeCamera window with Torsion ON





Arrington Research

The samples of iris striations are taken along an arc around the pupil. The radius of the SamplingArc is adjusted using the *Torsion window > Radius slider* that allows adjustment of the radius away from the edge of the pupil, the angular starting position of the arc (*Angle slider*), and the length of the arc (*ArcDeg slider*).

The user should adjust the arc to a location where there is good variation in the iris striations. Sometimes iris striations are periodic, like a sine wave; each striation looks the same to the software so measurement of rotation beyond the period of one oscillation is impossible. Irregular iris markings are required for accurate torsion measurement. The location of the SamplingArc is drawn in the *EyeCamera* window.



Any image feature that is not part of the iris (glints, eyelids and lashes, etc.) must be avoided.

The threshold dots are automatically turned off when using torsion. This is important because the threshold dots are painted in the video image before the torsion sample is taken and this can adversely affect the performance of the torsion calculation.

The amount of calculated torsion in degrees is displayed in the *Torsion* window, the *PenPlot* window and stored in the data file

11.2 Procedure for Measuring Torsion

This section describes the procedure for obtaining ocular torsion measurements using *ViewPoint*. This is an optional extra available only if the Torsion feature was purchased. It is assumed that the user is familiar with setup and thresholding as described in section [5.4](#).

To open the *Torsion window*, select: *EyeCamera window > Monitor Icon > Torsion*.

1. Ensure that *Auto-Set after adjust* is checked.
2. Press the [*START*] button on the *Torsion* window. The button label will not say [*STOP*].
3. Adjust the camera so that the video image of the subject's pupil is in the middle of the *EyeCamera* window and the iris is in sharp focus.
4. Adjust the brightness and contrast if necessary of the Threshold image.
5. Instruct the subject to fixate at a given point.
6. Use the *Radius*, *Angle*, and *ArcDeg* sliders to adjust the SamplingArc to a location where there is strong irregular variation in the iris striations.
7. Ensure that the SamplingArc does not include specular reflections, the eye lid, and any other non-rotating areas of brightness or shadowing.
8. Press the [*Set Template*] button, when the subject's eyes are at zero torsion.
9. Collect data as usual.



When the *Real-time graphics* check box is checked, the graphics windows are updated with every vertical refresh. If you need to reduce the computational burden, uncheck Real-time graphics. This will only update a few times per second. This does not affect the real-time data stored in the data file.

11.3 Torsion Demonstration Test

This section describes a simple test that can be useful for verification and for building intuition about how things are working. Normally the *Template* is reset whenever the SamplingArc location changes, but if we turn OFF this feature, then changing the *angle* of the SamplingArc will make the data show ocular torsion of the amount that the *Angle slider* was changed.

This starting point can be adjusted using the *Angle slider*. This can be used for testing and demonstrating the torsion calculation as follows:

- ④ Uncheck *Auto-Set after adjust*.
- ④ Adjust the sample angle using the *Angle slider*.

As the pattern shifts away from the template pattern, the correlation peak shifts and the torsion calculation changes.

When not performing this demonstration, *Auto-Set after adjust* should always be checked, since a new autocorrelation template will be required if the radius of the arc and angular starting point on the arc is adjusted. The autocorrelation template can also be re-fixed manually at any time by pressing the *Set Template* button.

Section [19.21](#) describes the torsion commands in detail.

11.4 Cyclovergence

[*Zero Sync*] button is used to simultaneously set the nominal zero position of the two eyes. It does not reset the templates. Note that there is no absolute position that is clearly zero degrees torsion. In general this would be done when the subject is looking straight ahead. The [*Zero Sync*] button is primarily important when we want to look at differences in the amount of torsion between the two eyes, called *cyclovergence*. The cyclovergence values and graph can be displayed in the *PenPlots* window.

The [*Set Template*] button will reset the sample of iris striation intensities that is used as the reference.

11.5 Overriding the Default Torsion Parameters

The default setting is for *ViewPoint* to look for torsion over +/- 20 degrees. Beyond this will cause a “*Range Error*” to be reported in the *Torsion* window. The default precision is 0.5 degrees of arc. These defaults are in place as a tradeoff between range of torsion measured and resolution due to the high computational burden of the calculations performed.

Since the eye does not normally rotate about the line-of-sight more than about 9 degrees there is usually no need to perform the auto-correlation past this range, because increasing the range increases the CPU load unnecessarily. There are some situations in which this range needs to be increased, such as when the entire head is rotated.

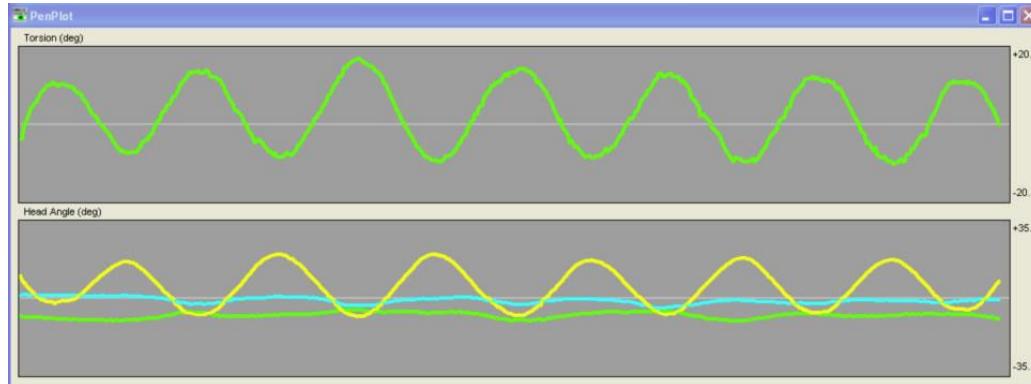
The user may adjust the torsion parameters with CLI commands; however, the user is responsible for testing that the combination they choose will provide valid results. Valid combinations should work up to +/-20 degrees at 0.5 resolutions.

11.6 Torsion Data

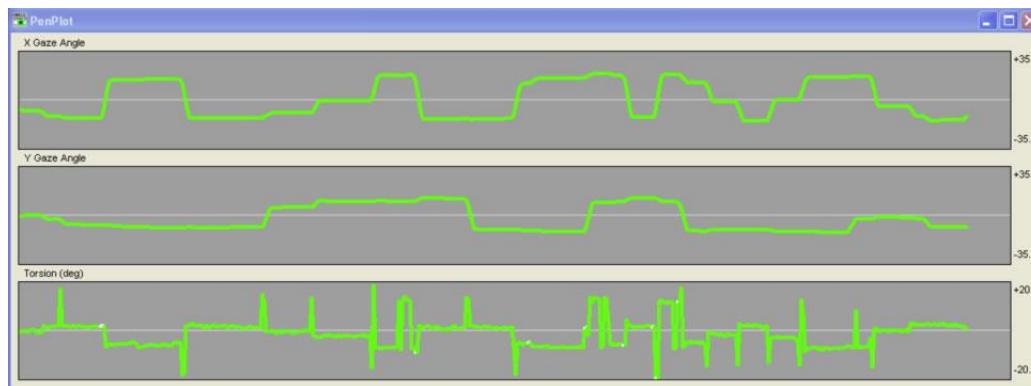
Ocular torsion occurs all the time as a normal part of eye movement. It allows the two eyes to align themselves so the rotational disparities between the two eyes are minimized. It helps stabilize the visual world in response to vestibular signals -- look closely at someone's eye as they tilt their head from one shoulder to the other and you should be able to see the torsional rotation of their eyes. It also occurs when looking in different directions; say up and to the right.

Figure 28. Examples of torsion data

Torsion from head roll:



Torsion from saccades:



Chapter 12. Stimulus Presentation (HeadFixed)

This section describes stimulus presentation options using the *ViewPoint* head fixed systems (e.g., with the *HeadLock* or *QuickClamp* products) system or a head mounted display (HMD) system.

12.1 Experimental Control

There are many ways to perform Stimulus presentation and Experimental Control with *ViewPoint*. The *Interfaces* section [2.2](#) describes the many ways that *ViewPoint* can be controlled with GUI, CLI, SDK, via Ethernet, etc.

ViewPoint interfaces with a wide variety of the most popular third party applications for experimental control and data analysis, as described in section [2.3](#).

ViewPoint itself can display images in *PictureLists*, play sounds, etc. using the built-in *StateEngine* for timing and control, as described in [Chapter 15](#).

12.2 Image Display

ViewPoint has integrated stimulus presentation capability. By selecting menu item:

File > Image > Load Image ... a stimulus picture (bitmap file) can be chosen using the standard open file dialog box. The picture will appear in both the *GazeSpace* window and in the *Stimulus* windows.

ViewPoint assumes that the BITMAP (.bmp) files are stored in the folder `~/ViewPoint/Images/`.

The user may specify an alternative default folder for image files, or may specify a full file path name when using CLI.

Hint 1: Make the bitmap image large so to avoid smooth lines being displayed as jagged.

Hint 2: Make the bitmap image the same aspect ratio as the *Stimulus* window.

The user has control over how images will be proportioned when they are displayed in the *Stimulus* window and *GazeSpace* window. By selecting menu item *Stimuli > Image Shape >*

- Actual Size*: the image will be displayed as actual size in the window.
- Center*: the image will be displayed as actual size and centered in the window.
- Stretch to Window*: the image will be scaled to fit the window.
- Stretch Isotropically*: the image will be stretched equally in all directions, maintaining the original proportions. Note that the *GazeSpace* window may automatically resize to match the image shape and matting shown in the *Stimulus* window.

Menu item: *Stimulus > Background Color* allows the user to change the background color in the *Stimulus* and *GazeSpace* windows. This is useful to provide “matting” color when the user has selected the image shape to be isotropic and there is space at the sides or at the top/bottom of the image.

12.3 PictureList

A *PictureList* is a list of picture file names. The file names may be loaded into the list using the *CLI* commands:

```
pictureList_Init  
pictureList_AddName myImageFileName-1_.bmp  
pictureList_AddName myImageFileName-2_.bmp  
etc.
```

This is typically done in a *Settings* file.

After being loaded, this *PictureList* may be randomized:

```
pictureList_Randomize
```

and then sequentially presented using, for example:

```
FKey_cmd 5 { pictureList_ShowNext }
```

Presentation may also be controlled with menu items:

File > Images > PictureList > Next PictureList Image

 presents the next image in the current *PictureList*.

File > Images > PictureList > Restart PictureList

 returns to the first item of the currently loaded *PictureList*.

File > Images > PictureList > Randomize PictureList

 randomizes the list of images in the currently loaded *PictureList*.

[Chapter 15](#) explains how the *StateEngine* can be used to very simply present a series of stimulus images at intervals determined by the user. A `pictureListEndAction` can be specified, as described in section [15.2](#).

Examples of how to use a *PictureList* can be found in the folder:

`~/ViewPoint/Settings/Examples/PictureLists/`

See section [19.6](#) for a complete list of *PictureList* commands.

12.4 Using the Stimulus Window (Head Fixed Option)

The *Stimulus* window is the window the subject sees. Calibration StimulusPoints and stimulus images are shown to the subject in this window. It is best when displayed full screen on a second monitor.

Use the [Controls window >Display tab](#) to remove or show the stimulus picture image. When unchecked the image is removed and all you see is the plain background color selected by the user.

[Stimuli > Stimulus Window Properties > Normal Adjustable Window](#)

Makes the *Stimulus* Window a resizable, moveable window.

[Stimuli > Stimulus Window Properties > Custom Static Position](#)

Sets the *Stimulus* Window to be a custom size as specified by the currently loaded *Settings* file. This window is not resizable or moveable. This feature is for use with non-standard display cards.

[Stimuli > Stimulus Window Properties > Full Screen Monitor 1 \(Primary\)](#)

Sets the *Stimulus* Window to be full screen on the primary monitor.

[Stimuli > Stimulus Window Properties > Full Screen Monitor 2](#)

Sets the *Stimulus* Window to be full screen on the secondary monitor.



To use a second monitor you will need to install a second display card or a card that supports multiple displays into your computer and configure your computer for multiple monitors. Please refer to your computer manual. Additional menu items will appear as more monitors are added.

[Stimuli > Stimulus Window Properties > Auto Show upon Calibrate \(toggle\)](#)

Automatically shows the *Stimulus* window and also automatically hides the cursor (only during calibration, re-present or slip correction) when the AutoShow occurs.

[Stimuli > Stimulus Window Properties > Auto Hide after Calibrate \(toggle\)](#)

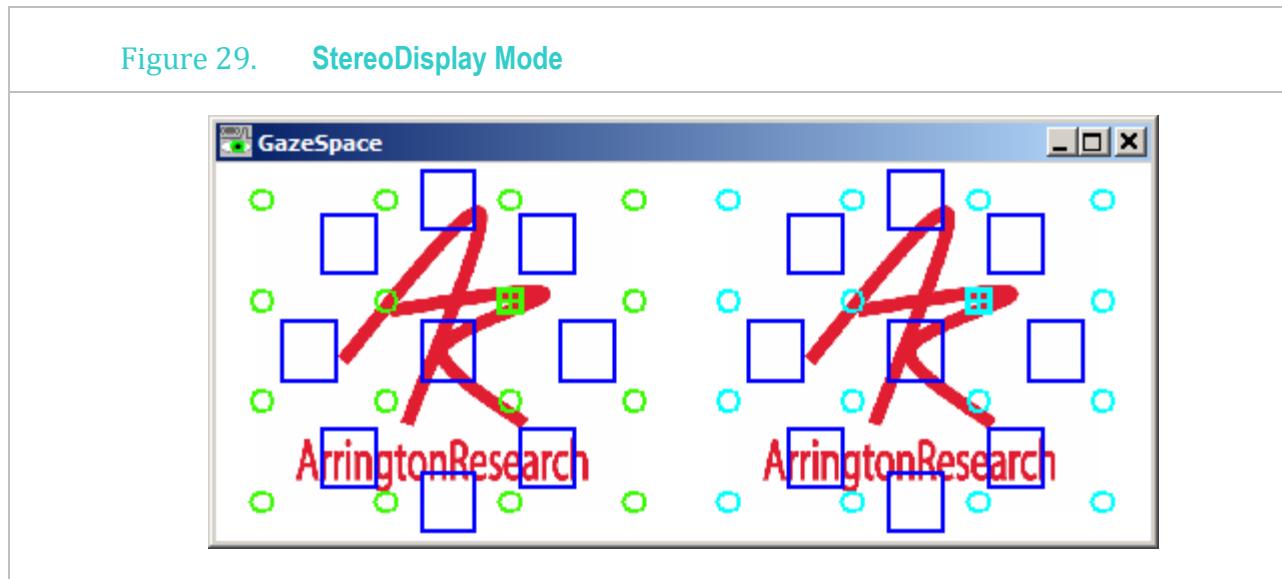
Automatically hides the *Stimulus* window after the calibration is complete.

12.5 Stereo Display

ViewPoint supports calibration and ROI with side-by-side 3D images (stereo images). Side-by-side has become the most common format. Selecting menu item:

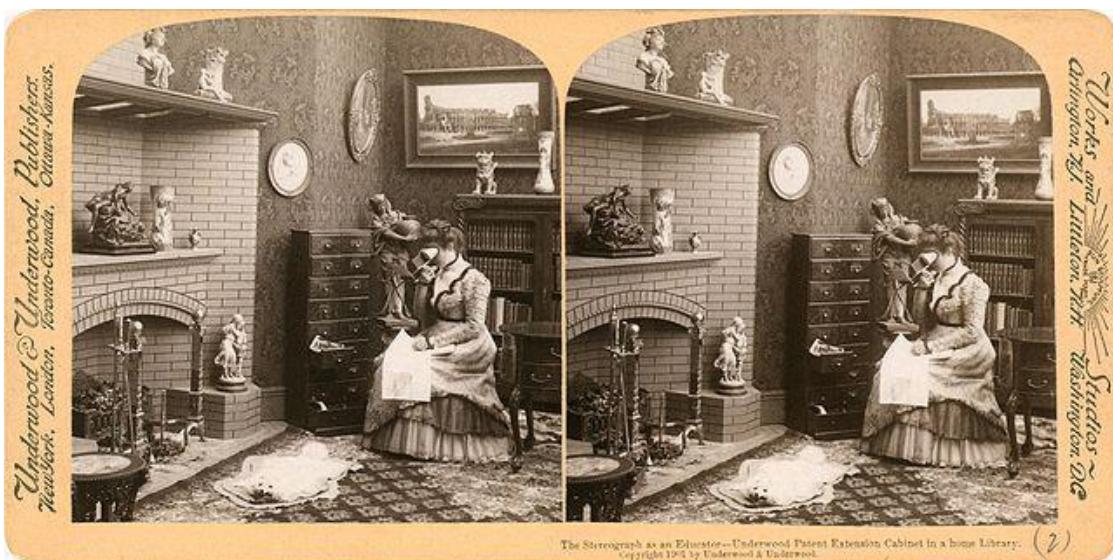
Stimuli > Stimulus Window Properties > Stereo Display (toggle), or issuing CLI command: `stereoDisplay ON`, causes the two color coded sets of calibration stimulus points and the ROIs to be separated for EyeA and EyeB, as shown in [Figure 29](#). *Binocular Mode* must be ON for the EyeB calibration points and ROI over EyeB to be shown. Use CLI: `stereoSwapEyes YES` to swap whether EyeA or EyeB is on the Left side.

See also [8.12.4.2](#) for custom positioning of calibration stimulus points and Partial Binocular Overlap (PBO).



Use CLI: `stereoPseudo ON` to tell *ViewPoint* to create a ‘fake-stereo’ or ‘pseudo-stereo’ image, by showing the same image on both the left and right sides of the display, as shown in the figure above. Note that in older *ViewPoint* versions, turning `stereoDisplay ON` had the side-effect of creating a pseudo stereo pair, but as of *ViewPoint* version 2.9.5.142, its behavior is controlled separately, which better accommodates real 3D stereo images, as shown in [Figure 30](#), below.

Figure 30. Side-by-side 3D Stereo Image



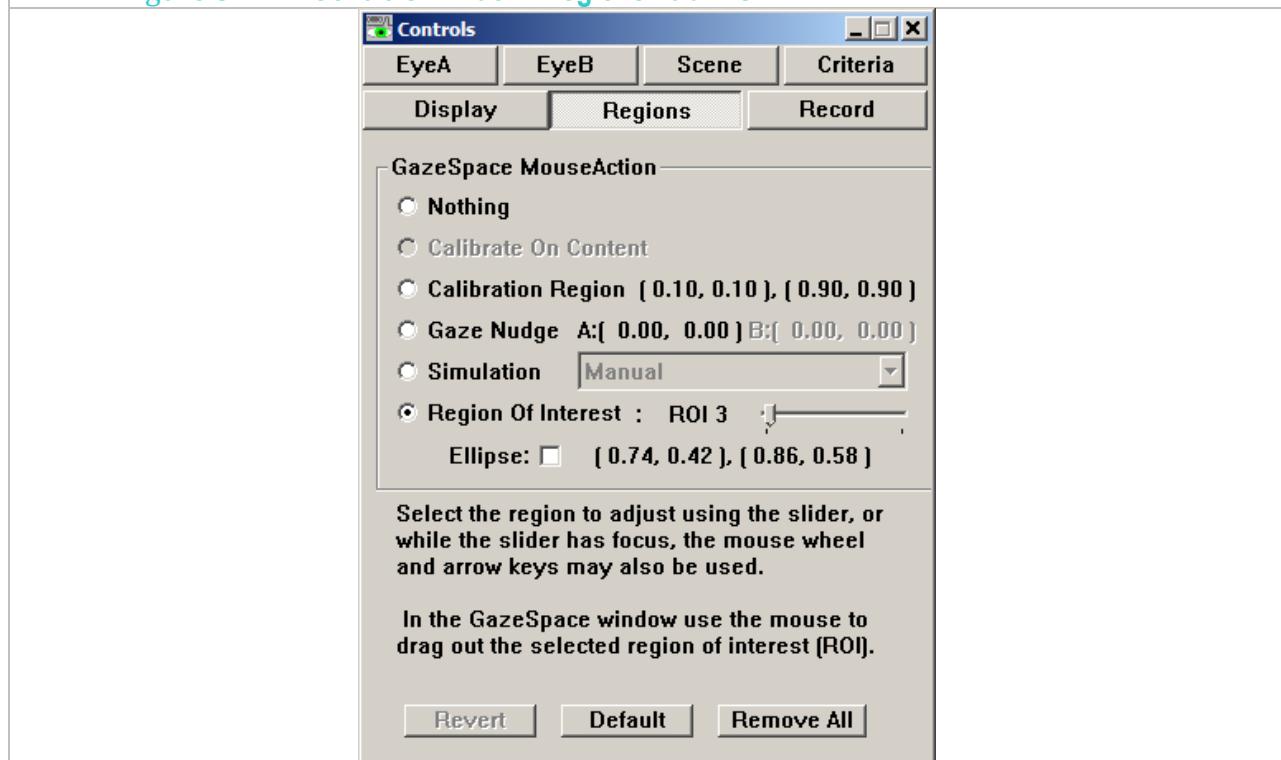
12.6 Regions of Interest (ROI)

The stimulus area can be divided up into *Regions Of Interest* (ROI), also sometimes called *Areas Of Interest* (AOI), or window discriminator boxes. These are very useful when the experimenter wants to know categorically whether or not the subjects gaze was in a certain area, which can simplify the task of data analysis.

It is possible to specify up to 99 ROI boxes (box numbers 1-99). When the gaze position moves inside a ROI, the ROI box number is stored in the data file record if a data file is open. If the boxes are overlapping and the gaze point is inside multiple boxes, then the data file will list all of the ROI boxes that were hit. The ROI that are hit are displayed in real-time in and above the *PenPlot window > ROI graph*. The *PenPlot* window shows the plot lines for ROI 1-7 and displays RIO numbers for all ROI that are hit that will fit in the data well above for that plot.

Note carefully that the *PenPlot* window shows the ROI event as soon as it starts, whereas the *Events* window shows the ROI event duration only after it is completed, that is, after the gaze goes out of the ROI, when the duration of the ROI event can be calculated, that is how long the gaze was inside the ROI. To adjust individual ROI, select the *Controls window > Regions tab > Region of Interest radio button*.

Figure 31. Controls Window: Regions Tab: ROI



Individual ROI may be selected by clicking the *right* mouse button *inside* the ROI. The selected ROI will be drawn in red and the others in blue. Drag out the bounding rectangle of the ROI while holding the *left_mouse* button down. Clicking the *right_mouse* *outside* of any ROI will set the adjustment mode to the *locked* state.



ROI can also be selected or locked using the *Controls window > Regions tab > ROI slider* to sequence through the RIO numbers. If the slider has focus you can also use the mouse wheel to sequence through the ROI numbers. The ROI coordinates of the selected ROI are displayed and updated in the *GazeSpace* window title bar and also next to the ROI slider, as you adjust the size and position of the ROI with the mouse. The shape of the ROI can be changed to an ellipse by checking the *Ellipse [x] checkbox* under the ROI slider.

The *Controls window > Regions tab > Revert button* will undo the last change and the *Default button* will return the ROI boxes to the start-up setting.

Use menu item *Remove All button* to remove all the ROI set.

12.7 ROI use Corrected Data

The eye trace lines displayed in the *GazeSpace*, *Stimulus* and *PenPlots* windows are *corrected* data. The corrected data can be smoothed to reduce noise, have parallax correction applied, binocular averaging, etc. (see section [14.2.1](#)).

Smoothing effects the real-time calculations. Because smoothing effects the velocity calculations the saccade velocity threshold must be adjusted proportionately. Smoothing will also affect which ROI boxes are triggered.

Note that *real-time* smoothing uses a trailing average technique, whereas *post hoc* data analysis should use a symmetrical smoothing technique. Both uncorrected and corrected data are stored in the data file.



For post-hoc analysis it is preferable to use a symmetric smoothing kernel on unsmoothed data.

12.8 Associating an Image with Specific ROI

You can associate an image with the ROI for that image. Every time an image is loaded, e.g. `~/ViewPoint/Images/MyImage01.bmp`;

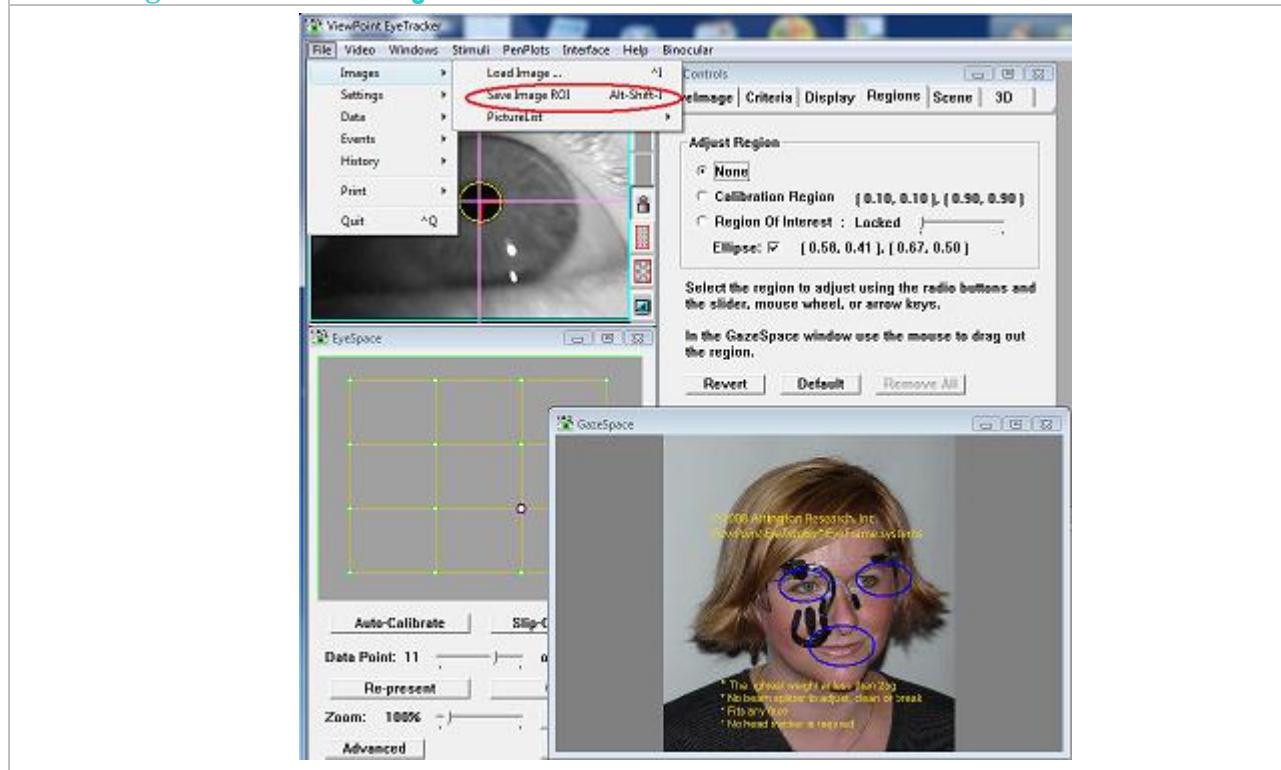
Whenever you load an *Image* file, *ViewPoint* also looks for and tries to load an *ROI file* in the same folder with the same file name plus the extension: `_ROI.txt`, for example the file shown above would have the following file associated with it: `~/ViewPoint/Images/MyImage01.bmp_ROI.txt` that contains the ROI specifications. The *ROI file* is just a special *Settings* file that is created using menu item: *File > Image > Save Image ROI* or CLI `savelImageROI`.

For example, you have a series of stimulus images of faces with ROI that include the eyes and mouth of each face. You want to present the stimulus images in turn and examine how long the subject spends looking in each ROI for each face.

Set up your ROI for each image and select menu item: *File > Image > Save Image ROI*

Setup *Picture Lists* as described in [12.3](#), above. Then, when the images are loaded into the *Stimulus* window and presented to the subject, the associated ROI are also loaded.

Figure 32. ROI Image Association

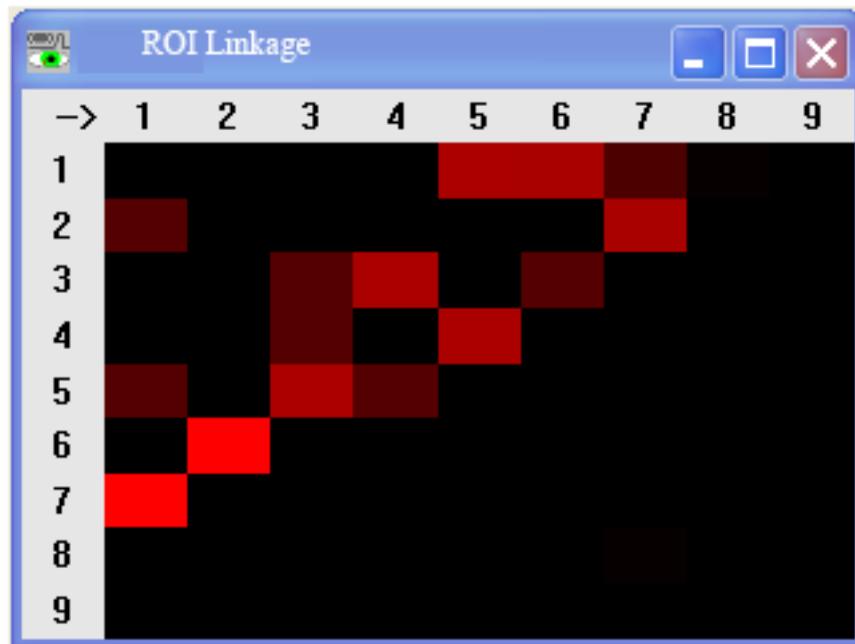


12.9 ROI Transition Statistics or Linkages

Transitions between different ROI are called *Linkages*. The linkage data can be viewed in a variety of ways.

The [ROI Linkage window](#) shows a graphical heat-map of ROI transition hot-spots. The values are first normalized, and then painted as intensities of red. It is interesting to watch this heat map as it is updated in real-time.

Figure 33. ROI Linkage Heat Map



The linkages can also be displayed in tabular form in the *History* window, which can be saved to file, as shown in the figure below. Use menu item: [Window > Dump Info > ROI Linkage Stats](#).

Figure 34. ROI Linkages Data

ROI Linkage:									
	--- To ---								
	1/	2/	3/	4/	5/	6/	7/	8/	9/
1]	3	4	1	2	1	2	2	1	0
2]	1	0	4	1	0	0	1	0	0
3]	0	1	0	5	0	0	0	0	0
4]	1	0	1	1	8	0	0	0	0
5]	0	0	0	1	0	9	1	0	0
6]	1	2	0	2	1	0	7	1	0
7]	1	0	0	0	1	3	1	8	0
8]	8	0	0	0	0	0	2	0	0
9]	0	0	0	0	0	0	0	0	0
<hr/>									
ROI#	Count	%Hits	SumDur	MeanDur					
1	15	0.17	1.34	0.09					
2	7	0.08	1.09	0.16					
3	6	0.07	0.74	0.12					
4	12	0.13	1.43	0.12					
5	11	0.12	1.16	0.11					
6	14	0.16	1.52	0.11					
7	14	0.16	1.45	0.10					
8	10	0.11	0.75	0.08					
9	0	0.00	0.00	0.00					
<hr/>									
Total:	89	1.00	9.48	0.11	<--- Overall				
For Help, press F1									

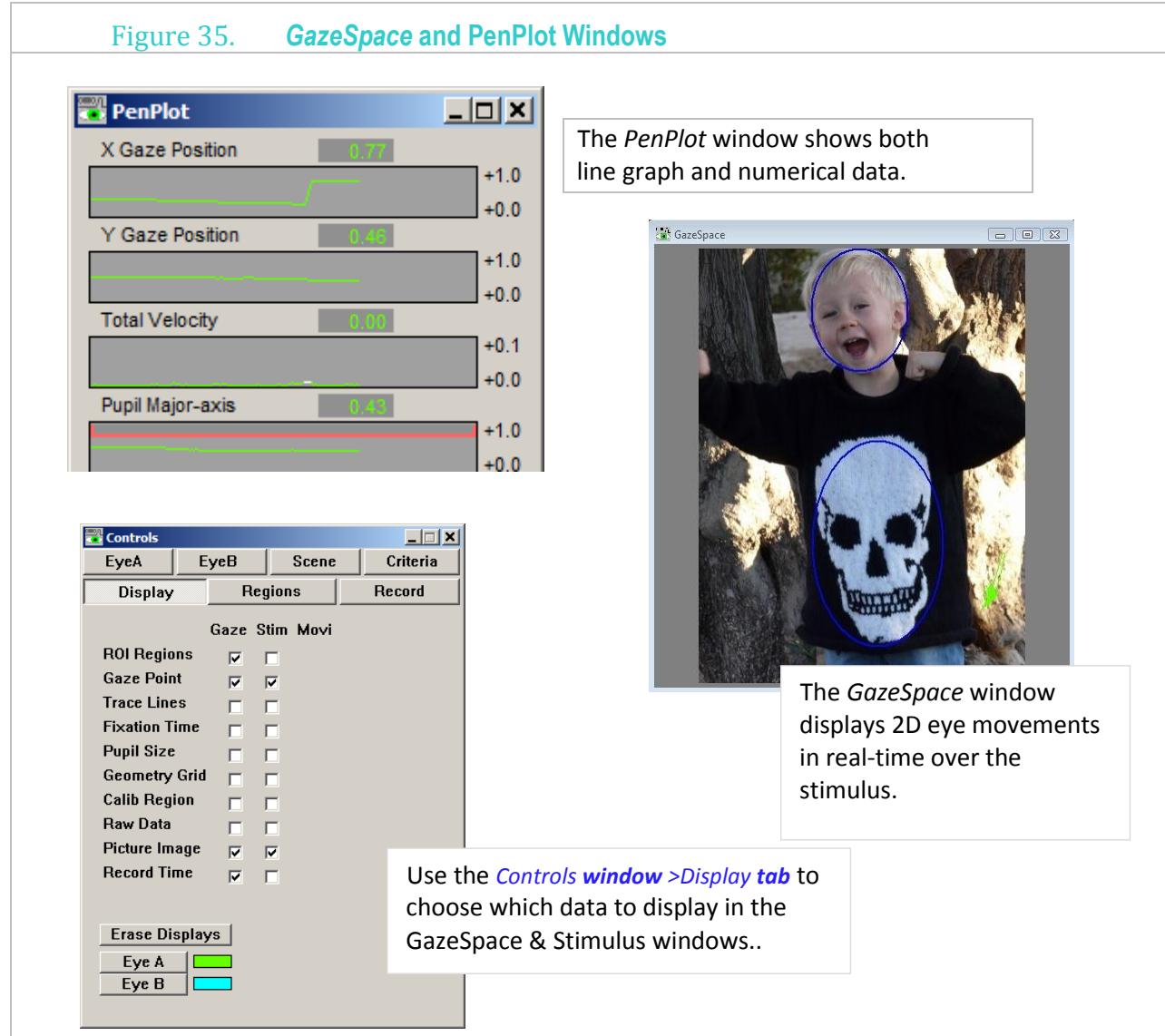
Careful inspection of the table will show that (a) the sequence must have started in ROI#1, because there are 16 transitions from ROI#1, but only 15 into ROI#1 and that (b) the sequence must have ended in ROI#5, because there are 12 transitions into it, but only 11 from it.

The “Count” values in the summary data is the sum of the “To” columns, rather than the sum of the “From” rows.

Chapter 13. Data Visualization & Analysis

13.1 Real-Time

The *ViewPoint EyeTracker*® provides several means of monitoring eye movements in real-time including the **GazeSpace**, the **PenPlot**, and the **Events** windows.



13.1.1 GazeSpace & GraphicsOptions

The *GazeSpace* window is a miniature representation of the *Stimulus* window. The experimenter may select from a variety of options to view the subject's eye movements and fixations. These can be set using the check boxes on the *Controls window > Display tab*, or using the CLI commands

```
gazeSpaceGraphicsOptions +graphicsOptions
stimulusGraphicsOptions +graphicsOptions
sceneMovieGraphicsOptions +graphicsOptions
```

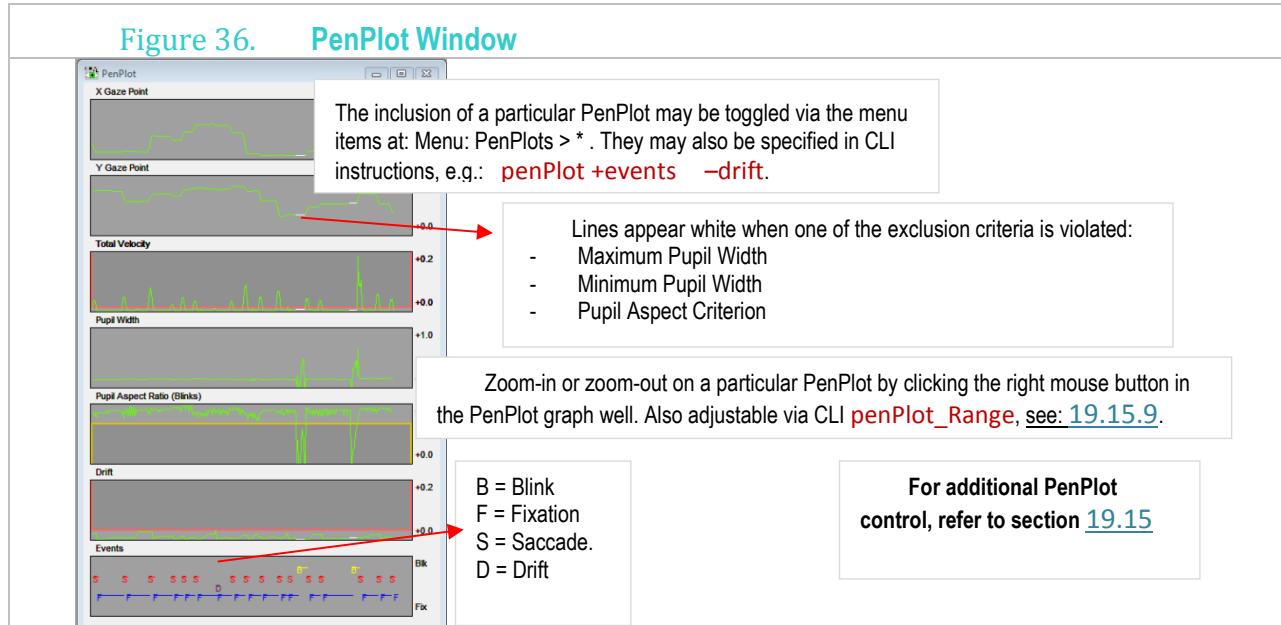
These *graphicsOptions* arguments are shown below in red, +option to add, -option to remove, ~option to toggle. See section [19.16.3](#) for more details. For example: `gazeSpaceGraphicsOptions +POG +Path`

- ④ *ROI Regions* displays the *Regions of Interest*. Change ROI under the *Regions* tab. `+ROI`
- ④ *Gaze Point* shows the subjects *Position of Gaze*. `+POG`
- ④ *Trace Lines* shows the path of eye movements. `+Path`
- ④ *Fixation Time* displays fixation duration as the increasing area of a circle. `+Fix`
- ④ *Pupil Size* will display a yellow oval corresponding to pupil size at the Position of Gaze. `+Size`
- ④ *GeometryGrid* to display the *GeometryGrid*. For *GeometryGrid* setup details refer to [\(9.1.5\)](#) `+Grid`
- ④ *Calib Region* shows the rectangle within which the calibration is performed, see section [8.12.6](#), and the locations of the calibration StimulusPoints when *calibration_PointLocationMethod* is set to *Automatic*. You can change the size and location of this rectangle in the *Controls* window > *Regions* tab > *Calibration Region* radio button. `+Calib` (NOTE: this is deactivated when *calibration_PointLocationMethod* is set to *Custom* or to *OnContent*, see section [8.12.4.1](#))
- ④ *Raw Data* to display the raw, uncalibrated data. `+Raw`
- ④ *Picture Image* to display the currently loaded stimulus picture image. `+Image`
- ④ *Record Time* to display the record time when in *SceneCamera ViewSource* mode. `+Time`
- ④ *Markers* to displays the Marker character at the current gazePoint location (Currently no GUI). `+Markers`. Markers can be inserted using CLI `dataFile_InsertMarker` command or with the *Controls* window > *Record* tab > *Marker* button.

These overlay graphics are usually only presented to the experimenter in the *GazeSpace* window, but they can also be presented to the subject in the *Stimulus* window. Display of eye traces in the *Stimulus* window is useful during setup and self- testing, but is not recommended during normal operation, since they can be very distracting to the subject.

13.1.2 PenPlots

The *PenPlot* window provides a wealth of real-time data. Use CLI: `penPlot_dumpNames` to get a complete list. See [Table 15, PenPlot names and Meanings](#) for more information.



13.1.3 Data Smoothing

Smoothing effects the real-time calculations of corrected data. Data displayed in the *PenPlot* and *GazeSpace* windows is corrected data.



Because smoothing effects the velocity calculations the saccade velocity threshold must be adjusted proportionately.

Smoothing will also affect which ROI boxes are triggered.

Adjust the amount of smoothing with the *Controls window > Criteria tab > Smoothing Points slider* or CLI `smoothingPoints`. When the slider is positioned at the far left, no smoothing is performed. Incrementing the slider to the right increases the number of previously obtained data points included in the average calculation. A value of 4 makes attractive and useful real-time graphics.

The smoothing algorithm can be selected using the *Smoothing Method pulldown menu*, or CLI `smoothingMethod`; in each case the number of *pointsBack* equals the number set by the user:

Simple Moving Average (**SMA**).

The SMA method uniformly averages N *pointsBack*, i.e., all points having equal weight.

$$\text{SMA}(t) = [x(t) + x(t-1) + \dots + x(t-n)] / N ; \text{ where } n = (N-1)$$

Exponential Moving Average (**EMA**).

The EMA method uses the following algorithm:

$$\text{EMA}(t) = (\text{currentValue} - \text{EMA}(t-1)) * K + \text{EMA}(t-1) ; \text{ where } K = 2 / (\text{pointsBack} + 1) .$$

Note that *real-time* smoothing uses a trailing average technique, whereas *post hoc* data analysis should use a symmetrical smoothing technique. Both uncorrected and corrected data are stored in the data file.

13.2 Fixation, Saccade, Drift and Blinks

13.2.1 Fixations

Three criteria values effect the creation of a completed *Fixation Event*: the *Saccade Velocity* criterion, the *Fixation Drift Allowed*, and the *Fixation Time Criterion*.

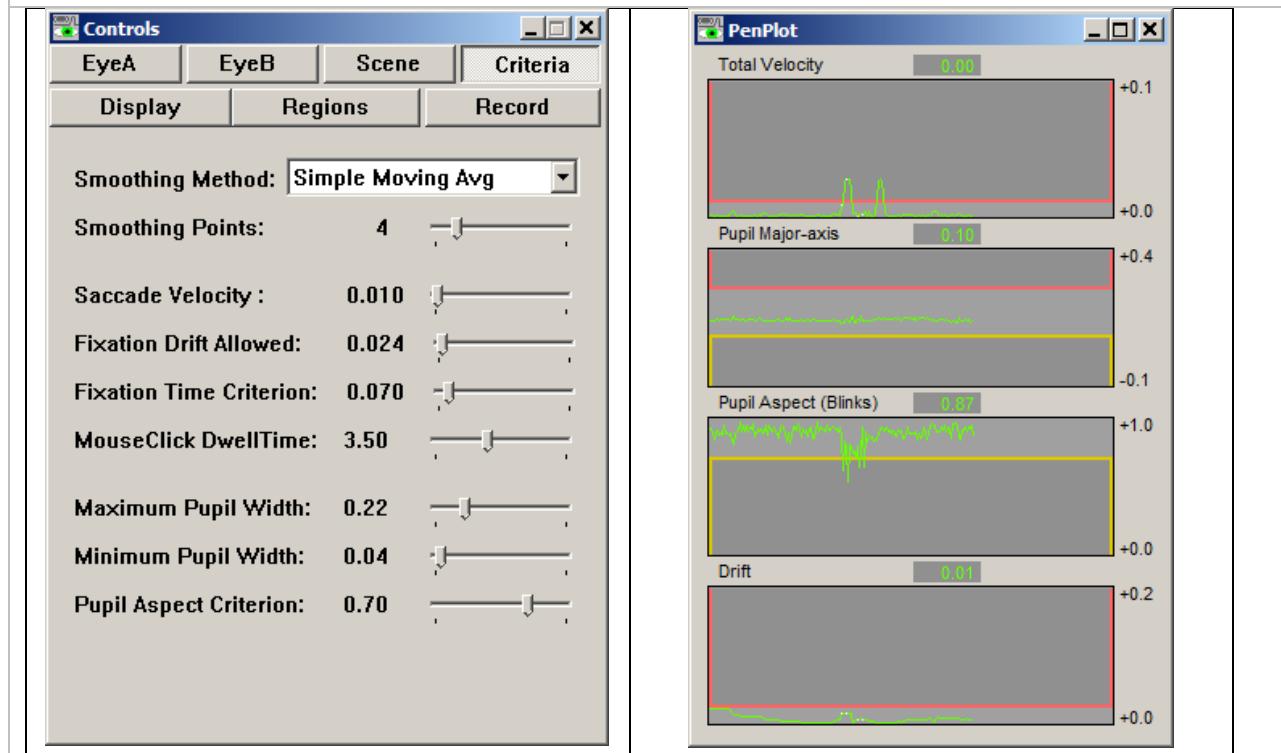
Whenever (a) the velocity was below the saccade velocity criterion, and (b) drift was less than the *Drift Criterion*, *ViewPoint* will start adding up fixation time. This fixation time is (a) included in the data file as a cumulative value, (b) displayed as a ramp function in the *Fixation Time* penPlot graphicsWell, and (c) marked in the *Events* penPlot graphicsWell by the letter '*F*'.

If the cumulative fixation time also meets the *Fixation Time Criterion*, then a completed *Fixation Event* is included in the *Events* window; this shows the average fixation location and the total fixation duration.

The fixation duration value appears in the data file in the **AFX** column. These are the same values as those shown in the *PenPlot* window in real-time. Note that these values are based on the smoothing algorithm, the amount of smoothing applied, and the qualitative saccade velocity threshold adjusted by the user (possibly in the middle of data collection). It may be preferable to apply a symmetrical smoothing kernel to the un-smoothed data during post-hoc data analysis and to determine the optimal saccade criteria at that time.

The following sections describe the fixation criteria in more detail.

Figure 37. Controls Window: Criteria Tab & PenPlot Threshold Lines



13.2.2 Criteria Levels

Criterion levels are displayed in the *PenPlots* window. Lower thresholds are distinguished by “legs” going down from the criterion level to the bottom of the penPlot graphicsWell. Upper thresholds are distinguished by “arms” going up from the criterion level to the top of the penPlot graphicsWell.

The upper and lower criterion levels are not allowed to cross, for example the *Minimum Pupil Width* cannot be greater than the *Maximum Pupil Width*, as shown in the *Pupil Major-axis* penPlot.

CLI: penPlot +TVelocity +Drift +FixationTime

13.2.3 Saccade Velocity Criterion

The *Controls window>Criteria tab>Saccade Velocity slider* (See [Figure 37](#)) provides a *qualitative* means for discriminating saccades from fixations. Adjustments to this threshold should be done while examining the *Total Velocity* penPlot graphicsWell in the *PenPlot* window. You can see when the subject fixates and saccades from the velocity spikes in this trace. The value of the spikes is simply the change in 2D position without being divided by sampling interval time.

GUI: *Controls window > Criteria tab > Saccade Velocity slider*

CLI: *velocityCriterion NormalizedValue*

Note that, changing the **Smoothing** value will affect the magnitude of the Total Velocity value.

Looking at the *PenPlot* when the subject is fixating you will see that the *Total Velocity* trace contains noise (thermal, video noise etc.) If saccades are large, then the placement of the saccade threshold is not so critical. If saccades are small more care should be taken. There is a trade-off in terms of misclassifying noise as a saccade or a small saccade being below threshold.

The *Saccade Velocity slider* value does not have the same affect on the EyeData after changing the frame rate or the *Stimulus* window size. This is a direct result of the Total Velocity being the change in 2D position, rather than a true derivative velocity (divided by time) and because the Total Velocity is in normalized units relative to the *Stimulus* window.

Ex 1: Let's say the POG travels at a constant speed and covers a 0.3 distance in 1 second. On a 30 Hz system, each eye frame would show a distance traveled of 0.01. On a 60Hz system, each eye frame would show a distance traveled of 0.005. So the magnitude of the Total Velocity is higher for the slower 30Hz frame rate.

Ex 2: Let's say you calibrate on a *Stimulus* window that is full screen and you gaze from the left side to the middle of the *Stimulus* window at a constant velocity in 1 second. The distance traveled would be 0.5. Now let's say you calibrate on a *Stimulus* window that is half full screen size and you gaze from the left side to the right side of the *Stimulus* window at a constant velocity in 1 second. The distance traveled would be 1.0. In the real-world, the distance traveled in 1 second was the same (half the monitor) for both calibrations but since the distances are normalized with respect to the *Stimulus* window they are not the same (0.5 vs. 1.0). So the magnitude of the Total Velocity is higher for the smaller *Stimulus* window.

This means that you may have to readjust the Saccade Velocity criterion when changing frame rates or when changing the size of the *Stimulus* window in order to trigger similar saccades as before.

The *3DViewPoint* and *3DWorkSpace* products provide velocity in degrees per second.

13.2.4 Drift Criterion

Note that the eye may be slowly drifting or in a low velocity smooth pursuit, such that the velocity is below the *Saccade Velocity* criterion. For this reason we also provide a *Drift* criterion that establishes the maximum distance that the POG may move away from the putative fixation point. That is, we impose the additional criterion that the POG must remain within a certain distance of the initial fixation point. Fixation drift is in units of normalized *Stimulus* window width.

GUI: *Controls window > Criteria tab > Drift Allowed slider*

CLI: `driftCriterion NormalizedValue`

The absolute drift distance and the specified Drift criterion level are shown in the *Drift* penPlot graphicsWell. When drift exceeds criterion level, the letter ‘D’ appears in the *Events* penPlot graphicsWell.

13.2.5 Fixation Time Criterion

Specifies the minimum fixation time required for a tentative fixation to be classified as a Fixation Event. This criterion level is displayed in the *PenPlot* window *Fixation Time* graphicsWell as a lower level line with vertical legs going down to the bottom.

GUI: *Controls window > Criteria tab > Fixation Time Criterion slider*

CLI: `fixationMinimumSecondsCriterion seconds`

13.2.6 Blinks

As the eye lid comes down during a blink, the elliptical fit to the pupil becomes increasingly flat before it disappears. This characteristic change in the aspect ratio of elliptical fit to the pupil can be used to detect blinks. A blink is classified as the pupil aspect ratio crossing below the threshold.

GUI: *Controls window > Criteria tab > Pupil Aspect Criterion slider*

CLI: `pupilAspectCriterion NormalizedValue`

13.2.7 Events Markers

Provisional event markers for Fixations, Saccade, Drift and Blink events are displayed in the *PenPlot* window in the *Events* graphicsWell as the following characters:

D – Drift

F – Fixation

S – Saccade

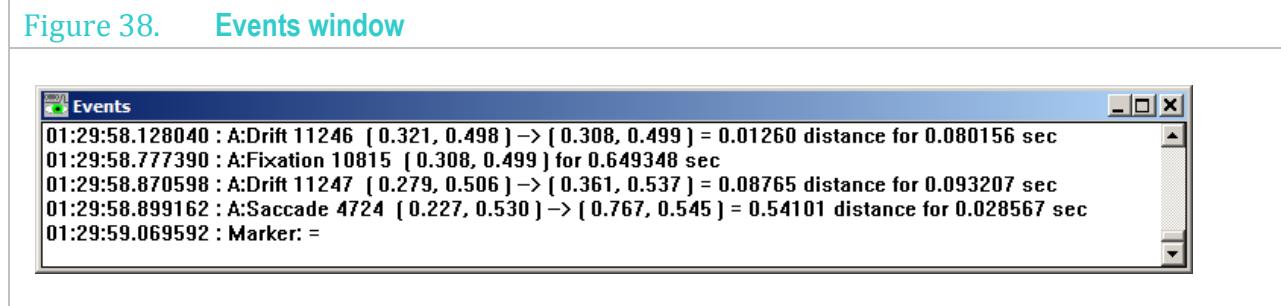
B – Blink

Fully qualified events are also displayed in the *Events* window after they end (because we need the duration of the event). Be careful to note that the fixation **F** will appear in the *PenPlot* window as soon as the fixation starts, but the completed fixation will not be posted to the *Events* window until it is completed, and only if it meets the Fixation Time Criterion, because the *Events* window shows the duration of the fixation.

13.2.8 Events Window

The contents of the *Events* window can be saved to either the **DataFile** or a separate **EventsFile**; see section [14.9](#) for details.

Figure 38. Events window



13.2.9 SDK

The real time values for all DataPoints are available via the DLL based SDK. See Section [20.9](#).

13.3 Post-Hoc

The data file format is described in [Chapter 13](#).

We also provide a basic data analysis program that allows displaying data in both a *LineGraph* (time plot) and a 2D (x,y) plane overlaid on the visual stimulus. The **DataAnalysis** program can be launched by itself from the folder or it can be launched when the data file is closed. See section [5.7, Figure 8](#).

Button: *Controls window > Record tab > Analyze button*

Menu: *File > Data > Close & Analyze Data File*

CLI: `dataFile_CloseAndAnalyze`

By default, *ViewPoint* looks for `~/ViewPoint/DataAnalysis.exe`, however this may be changed using CLI: `dataAnalysisApp`, for example:

```
dataAnalysisApp "anotherAnalysisApp.exe" // Defaults to the ViewPoint folder
dataAnalysisApp "C:/ARI/DataAnalysis.exe" // Full path
dataAnalysisApp "C:/Windows/write.exe" // Launches Wordpad
```



For post-hoc analysis it is preferable to use a symmetric smoothing kernel on unsmoothed data.

13.4 Summary Data

Summary data can be viewed using menu item: *Window > Dump Info > ROI Linkage Stats*. This includes fixation and blink summary data as well as ROI transitions. Refer to [Figure 34](#).

Chapter 14. Data Collection

14.1 Sampling Rate

ViewPoint includes various *modes of operation* that provide the user with predefined combinations of resolutions and sampling rates. For example, 320x240@60Hz, or 320x200@220Hz. Available modes and sampling rates depend upon the product purchased. Which mode you choose will depend on your research or project requirements. Description of these modes can be found under the various sections in [Chapter 24, Hardware Installation](#).

The *EyeCamera* window *Eyelimage* is always displayed at 320 X 240, or a subsection thereof, regardless of the mode selected.

14.2 Saving Data to File

The *ViewPoint* data file is always saved as tab-delimited text, sometimes referred to as the standard spreadsheet format, so it can be easily read by most any program, e.g. *OpenOffice Calc*, *LibreOffice Calc*, *Microsoft Excel*, *Microsoft Works*, *MATLAB*, *Mathematica*, etc. The file extension does not affect the internal format of the data file. You can specify, or change the file extension whenever, and to whatever you want (e.g.: `.txt`). You can associate the file extension with specific applications at the operating system level, so that when you double-click the file icon, a specific application will open it.

The data file will contain information about the (x, y) gaze point, elapsed time since the last entry, total time, pupil diameter, and a Region of Interest box number, etc. By default, the data files are stored in the folder named `~/ViewPoint/Data/`. The default folder can be change using CLI: `setPath`, described in Section [19.25.5](#).

To start recording data to file either:

Select menu item: *File > Data > New data file*.... This allows the user to open a data capture file, and to specify a file name using the standard *Open File* dialog box.

Select menu item: *File > Data > Unique data file ... (Ctrl-U)* This opens a new data capture file with a unique file name without having to go through the standard *Open File* dialog box. The default extension for these files can be set using the CLI: `dataFile_NewUniqueExtension` and substitutions specifiers (e.g. `%Y` for the current year) can be added using `dataFile_NewUniqueFormat` and `dataFile_SubstitutionPair`, described in Section [19.3.1](#). The file name created will include the date and time started starting with the year.

File > Data > Pause Data Capture (Ctrl-P) will temporarily stop the recording of data to file while processing continues. Data recording can be continued by toggling this menu item off. This feature is different from pressing the freeze button on the *EyeCamera* window which stops all processing including image capture, as well as the recording of data to file. *File > Data > Close Data File (Ctrl-W)* terminates the writing of data and closes the open file.

The Command Line Interface (CLI) for data files is described in Section [19.3](#).

14.2.1 Unprocessed & Corrected Data

Two types of Gaze space data are stored in the data file:

Unprocessed data is the calculated Position of Gaze, nothing more.

Corrected data includes any:

Smoothing



Binocular Averaging
Parallax Correction
GazeNudge Correction, etc.

Typically, you would apply these corrections post-hoc from the unprocessed data but, for convenience, we provide them for you. These corrections require additional setup and will cause the data to be incorrect if they are not done properly.

ROI, Fixation duration and events are based upon corrected data

14.2.2 AlsoMake

ViewPoint allows the user to specify that other files are to be opened and closed at the same time as the *DataFile* and they will all have the same *newUnique* core name. Precede the other file types with a + to add them or a - to remove them.

dataFile_AlsoMake +EyeMovie +Events +History

14.3 Data File Format

14.3.1 File Header Information

At the top of each data file is file header information that includes the date and time that the data was collected, the apparatus geometry settings that can be used to obtain the gaze angle in degrees, whether smoothed or unsmoothed data was stored, etc.

14.3.2 File Records

Each line of the data file is a unique data record. The type of record is indicated by the integer “**Tag**” value in the first column. The tag lets us know how to interpret the entries that follow on that line. The record entries on the line are tab separated into column positions. The number of columns in the eye-data record will depend upon the options selected for data collection. For example, if torsion is not being measured there will be no such column in the data file. If running in binocular mode, then additional data will be included for the second eye. The meaning of the various data record tags is described in detail in Section [14.3.4](#).

14.3.3 Synchronous vs. Asynchronous Data Inserts

ViewPoint provides for synchronizing eye movement data with other events and other data. This is usually performed by inserting extra data into the *ViewPoint* data file. This extra data can include Markers, and Strings. These can be inserted in two ways, either (a) on the same line as the eye tracker data (synchronously), which makes reading into spreadsheets much easier, or (b) on separate data lines (asynchronously), interleaved with the eye tracker data, which allows individual time stamps for each inserted item.

A simple method of synchronization with other devices and programs is achieved through insertion of ASCII character markers into the data stream. By default, menu item *File > Data > Asynchronous Marker Data* is toggled OFF, so data markers will be synchronously added into the data file stream, in the Marker column. If more than one character was inserted, they will appear as a comma separated list of characters. If this menu item is checked ON, ASCII character data markers will be asynchronously inserted into the data file stream as they arrive, with individual time stamps, and there will be no Marker column.

By default, menu item *File > Data > Asynchronous String Data* is toggled ON and string markers will be asynchronously added into the data file stream with individual time stamps. See [Figure 39](#).

Figure 39. DataFile with Synchronously and Asynchronously Inserted Markers

Run the following set of CLI commands from a *Settings* file:

```
dataFile_asynchStringData      YES
dataFile_InsertString         "CAT_A (asynchronous)"
dataFile_asynchStringData      NO
dataFile_InsertString         "DOG (synchronous)"
```

Produce the following data file lines:

```
10  26.5228  16.6076  0.6750  0.2083  0.4598  0.3208  -1  0.0161  0.0161  1  0.0000  315
12  26.538822  CAT_A (asynchronous
10  26.5394  16.6255  0.0406  0.4292  0.2922  0.3641  -1  0.0161  0.0161  1  0.0000  316  DOG (synchronous)
```

14.3.4 Data Record Tags.

The “**Tag**” value that begins each line in the first column indicates the type of information in that line, i.e., in that data record. Details are provided in the tables here below; the primary record types are as follows:

Tag #2: Marker: Asynchronous ASCII Marker Character. If menu item *File > Data > Asynchronous Marker Data* is checked ON, then single ASCII character event markers will be inserted asynchronously into the data file. Certain ASCII characters are automatically entered into the data stream to indicate a particular event has occurred. The type-2 record contains three entries, as described in [Table 8](#).

Tag #3: Header Information: An ASCII character string generated by *ViewPoint* to provide general information, such as when the data file was created.

Tag #5: Data Column Heading: An ASCII character string generated by *ViewPoint* to provide column heading information. These are for easy understanding by the user. Refer to Table below.

Tag #6: Data Column Identifier: A three character data column identifier generated by *ViewPoint*. These are for analysis, such as by the `DataAnalysis.exe`. Refer to [Table 11](#)

Tag #10: EyeData: EyeData containing a variable number of columns depending on the options selected for data collection. Refer to Table 6.

Tag #12: InsertString: An ASCII character string asynchronously inserted during certain events. For example, by CLI commands : `dataFile_InsertString "picture of a cat"` from a *StateEngine StateEntryCommand* or *StateExitCommand*, from a *Settings* file, or from an external program. Refer to [Table 10](#).

Tag #14: HeadTracker data: Asynchronously inserted.

Tag #16: StimulusImage: Picture File Name.

Tag #111: UserData, see section [14.3.5](#).

Tag #777: Movie: MovieType MovieFrameNumber.

Tag #800-899: User Defined Tag: `dataFile_InsertUserTag`, see section [14.3.6](#)

14.3.5 UserData

ViewPoint accepts real-time real number (floating point) UserData from external sources, displays it in the *PenPlot* window, and saves it asynchronously as it arrives into the *DataFile*.

The *UserData* is saved into DataGroups. Each DataGroup can contain up to six DataStreams that are displayed in a single penPlot graphicsWell. For example:

```
userData 2 1.1 2.2 3.3 4.4 5.5 6.6 // Sets new values for six dataStreams in Grp#2
```

These six values are all displayed in a single penPlot graphicsWell as separate pens of various colors. To display these dataStreams in dataGroup *userData2*, use the CLI command:

```
penPlot +userData2 // Show penPlots of UserData2 group data.
```

You can also specify the range of data displayed in the penPlot graphicsWell as follows:

```
penPlotRange userData2 -12 +12 // Sets the range of data shown.
```

ViewPoint currently allows six dataGroups. Each dataGroup can contain up to six dataStreams.

ViewPoint will only display the number of dataStreams actually used, for example it will only show three pens in the penPlot graphicsWell in the following example:

```
userData 2 1.1 2.2 3.3 // Specifies new dataValues for three dataStreams.
```

The number of dataStreams in a dataGroup is automatically detected. If you decide to send fewer dataStreams in the middle of a work session, you can reduce the number of pens shown in the penPlot graphicsWell with the following CLI command:

```
userData_ResetAutoDetect // Only activate dataStreams actually used
```

The *UserData* is stored in the *DataFile* asynchronously as it arrives as follows. The tag is 111, the *TimeStamp* is next, then the DataGroup number (1 in the example here below) then the current data value in the first DataStream (360 in this example) of the group, etc.

10	2.7656	16.6577	0.9594	0.5875	0.6389	0.5113	7	0.0161	0.0161	5	0.0000	166
10	2.7822	16.6292	0.6896	0.3429	0.6592	0.4440	7	0.0226	0.0158	1	0.0333	167
111	2.799901	1	360	222	3.3	4.4	5.5	6.6				
10	2.7988	16.6349	0.6531	0.5000	0.6568	0.4664	7	0.0161	0.0161	1	0.0499	168
10	2.8153	16.4533	0.6531	0.5000	0.6568	0.4664	7	0.0161	0.0161	5	0.0499	169
10	2.8321	16.7804	0.8906	0.3708	0.7503	0.4282	-1	0.0161	0.0161	1	0.0832	170
10	2.8488	16.7133	0.8719	0.7333	0.7989	0.5502	4	0.0161	0.0161	1	0.0167	171
10	2.8655	16.7758	0.8719	0.7333	0.7989	0.5502	4	0.0161	0.0161	5	0.0167	172
10	2.8822	16.6153	0.9375	0.4417	0.8544	0.5068	4	0.0161	0.0161	1	0.0334	173
10	2.8988	16.6606	0.9375	0.4417	0.8544	0.5068	4	0.0161	0.0161	5	0.0334	174
111	2.909942	1	360	222	3.3	4.4	5.5	6.6				
10	2.9153	16.4793	0.3016	0.4199	0.6332	0.4720	7	0.0207	0.0164	1	0.0000	175
10	2.9318	16.5471	0.8937	0.6458	0.7374	0.5416	-1	0.0161	0.0161	1	0.0000	176



14.3.6 User Defined Tags

Users can insert data from their own sources (EEG, EMG, 6-DOF trackers, etc.) into a data file using their own specified data tag. The tag appears in column 1 of the data file. The insertion is done asynchronously with respect to the eye movement data records and the insertions are uniquely time stamped. Tag identifiers must be in the range 800-899. Use the CLI: `dataFile_InsertUserTag` . See section [19.3.7](#).

Unlike the *UserData* in the previous section, this information is not necessarily numeric and is not displayed in the *PenPlot* window.

Table 7. Eye Data

Column Heading	Type	Description
Tag	int	The value 10 in the first column indicates an eye data record
TotalTime	float	TotalTime = time elapsed in seconds
DeltaTime	float	dt = delta time in milliseconds since the previous data entry
X_Gaze	float	X = Direction of Gaze normalized with respect to the x-axis
Y_Gaze	float	Y = Direction of Gaze normalized with respect to the y-axis
X_CorrectedGaze	float	Same as X_Gaze but may include smoothing, averaging, parallax correction, nudging, etc.
Y_CorrectedGaze	float	Same as Y_Gaze but may include smoothing, averaging, parallax correction, nudging, etc.
Region	list	Which ROI or ROIs the gaze point is in
PupilWidth	float	Pupil width normalized with respect to the <i>EyeCamera</i> window width. See section 14.4 .
PupilHeight	float	Pupil height normalized with respect to the <i>EyeCamera</i> window width. See section 14.4 .
Quality	int	Quality of eye movement data. See section 14.11 for a complete description of the codes.
Fixation	float	Fixation duration in seconds. A zero value indicates a saccade.
PupilDiameter	float	Pupil Diameter is calculated using the major-axis (max dimension of PupilWidth and PupilHeight.) The normalized units of the pupil size are converted into millimeters using the pupil scale factor from the pupil calibration. See section 9.3 .
Torsion	float	Torsion in degrees. -998 indicate Torsion not being calculated. -999 indicate "Range Error". Only displayed if torsion is being measured.
Count	int	Eye movement data record count, useful for sorting
Mark	char	Any printable ASCII character, e.g., {a-z, A-Z, 0-9,=,#,+,%}, etc.}. See Table 8 , below.
pXraw	float	RawPupil. Only displayed if option to save raw data is checked
pYraw	float	RawPupil. Only displayed if option to save raw data is checked
gXraw	float	RawGlint. Only displayed if option to save raw data is checked
gYraw	float	RawGlint. Only displayed if option to save raw data is checked

Table 8. Asynchronous Marker Record Structure, Tag = 2

Column #	Type	Value
1	integer	2 = integer indicates data structure type 2



Arrington Research

2	float	Time Stamp
3	character	Any ASCII character, e.g., {a-z, A-Z, 0-9, =, #, +, %, etc.} Automatically inserted tags include: + Start or resume data collection, and = Pause data collection

Table 9. String Data Record Structure, Tag = 3

Column #	Type	Value
1	integer	3 = integer indicates data structure type 3
2	string	File header information, an ASCII character string generated by <i>ViewPoint</i>

Table 10. String Data Record Structure, Tag = 12

Column #	Type	Value
1	integer	12 = integer indicates data structure type 12
2	float	Time stamp
3	string	ASCII character string from "dataFile_InsertString", etc.

Table 11. Column Header Data Record Structure (Tag # 5 and Tag # 6)

Column #	Type	Tag Value		Eye / Head
1	Integer	5	6	
2 – n	String	TotalTime DeltaTime X_Gaze Y_Gaze X_Corrected Gaze Y_Corrected Gaze Region (ROI) PupilWidth PupilHeight Quality Fixation PupilDiameter Torsion	ATT ADT ALX ALY ACX ACY ARI APW APH AQU AFX APD ATR	Eye_A
		TotalTime DeltaTime X_Gaze Y_Gaze X_Corrected Gaze Y_Corrected Gaze Region (ROI) PupilWidth PupilHeight Quality Fixation PupilDiameter Torsion	BTT BDT BLX BLY BCX BCY BRI BPW BPH BQU BFX BPD BTR	Eye_B
		X Y Z Yaw Pitch Roll	HPX HPY HPZ HAY HAX HAZ	Head Tracker Option only
		Count Marker	CNT MRK	

14.4 PupilWidth and PupilHeight calculations

PupilWidth and *PupilHeight* calculations are relative to which pupil segmentation method is active.

Ellipse - *PupilWidth* is the major-axis (larger value) and *PupilHeight* is the minor-axis (smaller value).

The major-axis and the minor-axis are both normalized by the width of the *EyeCamera* window so the scales of the axes are commensurable.

Oval Fit - *PupilWidth* is the HORIZONTAL dimension and *PupilHeight* is the VERTICAL dimension of the rectangular bounding box of the unrotated Oval. The HORIZONTAL and VERTICAL dimensions are both normalized by the width and height of the *EyeCamera* window respectively so the scales of the axes are incommensurable.

Centroid - *PupilWidth* and *PupilHeight* are 0, because no pupil size is obtained.

14.5 Direction-of-Gaze Coordinates

The values **X** and **Y** are the coordinates of the direction-of-gaze with respect to the *Stimulus* window coordinate systems. For example, 0.0, 0.0 will mean that the Position of Gaze is in the top left hand corner of the *Stimulus* window; 0.5, 0.5 will mean that the Position of Gaze is in the center of the *Stimulus* window; and 1.0, 1.0 means that the Position of Gaze is in the bottom right hand corner.

The calculated (x, y) gaze location is initially in the same space as the calibration point locations. The calculated gaze is then normalized with respect to the x-axis and y-axis respectively. *ViewPoint* allows the GazePoint to be extrapolated outside of the calibration window. This is particularly important for using the *CursorControl Feature* (see [Chapter 10](#) Cursor Control Feature).

14.6 Raw Data

Raw (unmapped *EyeSpace*) data can be saved to the data file by selecting menu *File > Data > Include Raw (unmapped EyeSpace) Data*, or using CLI: `dataFile_includeRawData YES`.

14.7 Timing Measurement

ViewPoint includes high precision timing (HPT) with resolution in the order of 0.0000025, i.e., 2.5E-6 or 2.5 microseconds. The HPT is available to integrators via the SDK function call:

`VPX_GetPrecisionDeltaTime`. Refer to section [20.6](#) for details.

The Total Time and Delta Time values available in the DLL and the Data file are software time stamps. This means that *ViewPoint* puts a time stamp on the data when it arrives and the variability of the time stamps are due to CPU load handling. When hardware digital camera or frame grabber manufactures do not supply actual hardware time stamps, we use the software based time stamps.

Derivative calculations using time, like velocity, should assume a fixed delta time to prevent unwanted noise from the software variability of the time stamps.



The crystal clock in the camera is very precise, so the field and frame delivery from the camera is very regular. For example, with analog cameras frames are delivered at 16.6667 milliseconds for 60 Hz data, and 33.3333 milliseconds for 30 Hz data, respectively. Variability of the time stamps is due to CPU load handling.



In the *DataFile* the Total Time is the elapsed time in seconds starting with the first record of data collection. By default Menu item *File > Data > Start Data File Time At Zero* is checked (ON), which causes the first record of the data file to start at time zero. If this menu item is unchecked (OFF), the data records will start at the current High Precision Time maintained in the DLL and sharable between applications using the *ViewPoint* SDK via that DLL. See section [20.6](#).

To make the time more intuitive and more manageable, *ViewPoint* provides times since it was launched, rather than from when the computer was started.

Delta Time is the number of milliseconds since the previous matching record entry. This represents the time that the software has finished processing each frame and the eye movement data is available.

14.8 Display Screen Geometry

The data file also contains header information that specifies the results of the *GeometryGrid* calculations for screen size and viewing distance. The screen size values saved will be the values calculated for the entire screen width and height, not the measured lengths of the lines (that are 20% less) entered by the operator. See section [9.1.5](#).

14.9 Events Data

The content of the *Events* window can be included into the *DataFile* by checking the menu item: *File > Data > Include Events in Data File*, or using CLI: `dataFile_includeEvents YES`.

The content of the *Events* window can also be saved into a separate *EventsFile* using the menu item: *File > Events > New Events File ...*, CLI: `eventsFile_NewName`, or using the menu item: *File > Events > New Unique*, CLI: `eventsFile_NewUnique`, and `eventsFile_Close`.

More information on the *Events* window is in section [13.2.8](#)

14.10 Regions of Interest (ROI)

Often one is only interested in whether or not the POG is within specific regions. Regions of Interest (ROI), sometimes called Areas of Interest (AOI), may be defined as described in [12.6](#). When the corrected POG is within one of these boxes, the region number is stored in the data file. The ROIs may be overlapping or nested, and so the POG can be in more than one region at a time. The data file stores the current regions the POG is in at the time each *EyeData* record is processed. The available values for the Region column are:

- 1 indicates that the Position of Gaze (POG) is not in any of the regions,
- n indicates that the POG is in region n,
- n,m indicates that the POG is in more than one, overlapping ROI.



ROI #0 is deprecated, because it cannot be signed as +/- to indicate ROI into and out-of events respectively. Do not use ROI#0.

14.11 Quality Marker Codes

ViewPoint provides a data quality code for each eye. The *Quality* column contains an integer code ranging from 0, which is the best possible case, to 5. The hierarchical code structure allows the user to make their own assessment of whether the data is valid or not for their purposes. *Quality* marker codes are listed below:

Table 12. Quality Codes

Code	Description	
0	The user has selected to use the <i>glint-pupil</i> vector method and both features are successfully located.	VPX_QUALITY_GlintIsGood
1	The user has selected to use the <i>pupil only</i> method and the pupil was successfully located	VPX_QUALITY_PupilOnlyIsGood
2	The user has selected to use the <i>glint-pupil</i> vector method and the glint was not successfully located but the pupil was successfully located. Defaults to <i>pupil only</i> method for data recorded.	VPX_QUALITY_PupilFallback
3	In either the <i>pupil only</i> or <i>glint-pupil</i> vector method, the pupil exceeded criteria limits set.	VPX_QUALITY_PupilCriteriaFailed
4	In either the <i>pupil only</i> or <i>glint-pupil</i> vector method, the pupil could not be fit with an ellipse.	VPX_QUALITY_PupilFitFailed
5	In either the <i>pupil only</i> or <i>glint-pupil</i> vector method, the pupil scan threshold failed.	VPX_QUALITY_PupilScanFailed

[Section 20.4: Data Quality Codes](#), shows how a layered application uses the SDK to get quality values.

[Section 19.27.5](#) shows how the quality level can trigger TTL output changes.

Chapter 15. State Engine

ViewPoint contains a powerful state space engine for experiment control. The user defines a series of states which contain:

- `stateMode`: a unique numeric identifier for the state
- `stateLabel`: descriptive text to describe the state
- `stateEntryCommand` or `stateExitCommand`: a deferred CLI command or commands in braces, e.g.: to load an image, calibrate or to load a *Settings* file.
- Transition: `stateTimeout` action or, `stateJump`

15.1 State Engine Commands

15.1.1 State Initialization

The *StateSpace* must be initialized before using or reusing it. Command `stateSpaceInit` clears information for all states, sets all values to zero etc.

`stateEngine <boolValue>` turns the state engine on or off.

After turning the state engine on you must direct it to your chosen start state. `stateJump <integer nextState>` forces a jump to the specified state. If no state number is specified, then the engine will jump to state 0.

`stateRecordInit <int>` clears all information for the specified state and may be useful for debugging.

15.1.2 State Setup

Each state must be given a unique numerical identifier using `stateMode <int>`

The user can also specify a descriptive identifier for each state using: `stateLabel <string>`, e.g.

```
stateMode 1 // declares state 1
stateLabel "performs calibration"
```

A descriptive `stateLabel` is useful for clarity and is displayed when the command `stateDump` is called which dumps current state information to the *History* window.

15.1.3 State Commands

Each state may contain instructions to be sent to the CLI parser upon entry or exit from the state. Multiple commands are separated by semi-colons.

```
stateEntryCommand { dataFile_NewUnique }
stateExitCommand { dataFile_Close }
```

This can also include an instruction to load a separate *Settings* file, *Image* File, e.g.:

```
stateEntryCommand { pictureList_ShowNext }
stateEntryCommand { settingsFile_Load "experiment2/slipCorrect.txt" }
stateEntryCommand { stimulus_LoadImageFile "blue.bmp" ; midiNote 5 6 4 }
```

15.1.4 State Transition Commands

Each can include an action that triggers a jump to another state as follows:

Table 13. State Transition Commands	
Timeouts	stateTimeOut <double float seconds> <integer nextState> After the specified time has elapsed, then the state engine will jump to the declared next state.
ROI-Into ROI-OutOf events	stateEnterROI <integer roi index> <integer nextState> If the Position of Gaze enters the specified Region of Interest (ROI), then the state engine will jump to the declared next state. stateExitROI <integer roi index> <integer nextState> If the Position of Gaze exits the specified Region of Interest (ROI), then the state engine will jump to the declared next state.
Key presses	stateKeyPress <printable ASCII character> <integer nextState> If the user presses the keypad key for the specified character, then the state engine will jump to the declared next state. These are case sensitive and state dependent, i.e., they will only work when the engine is in that particular state. Note: For a global key press, use the FKey definition in the initialization part of your state file. E.g. <pre>FKey_cmd 9 { stateJump 2 } // when the user presses FKEY 9 then jump to state 2 // which presents the next image</pre>
Immediate Jump	stateJump <integer nextState> Forces jump to the specified state.

15.1.5 State Space Debug

stateDump prints detailed info to the *History* window.

15.2 Picture Lists

The user can define a series of images to be presented using the *PictureList* feature. Refer to [0](#)
 An end action can be defined that will be performed when the last picture in the picture list has been presented, i.e. the list pointer is incremented to the end. These do not need to be associated with a particular state, but can be defined in the initialization part of the file, for example:

State command: **pictureList_EndAction** string

E.g. { **pictureListEndAction** "stateJump 0" }

Or

pictureList_EndAction "dataFile_close"

etc.

15.3 Counters

Counters are useful when you want something to occur only after a certain number of other things have occurred (`counterZeroAction`), or every *N*th time some other thing occurs (`counterModulusAction`). Counters can be setup with an initial value (`counterSet`) and the incremented or decremented (`counterAdjust`). Counters are especially useful with the *StateEngine*, for example counter actions can be executed when a State is entered (`stateEntryCommand`) or exited (`stateExitCommand`).

1. `counterSet` *counterId* *countValue*

counterId is in { 0 ... 9 } a user defined identifier for that counter

countValue in any non-negative number { 0 ... maxInt } a user defined starting number for that counter. For example, to set the value of counter 0 to be 25 do: `counterSet 0 25`

2. `counterAdjust` *counterId* *adjustValue*

counterId is in { 0 ... 9 } a user defined identifier for that counter

adjustValue is any number { -maxInt ... +maxInt } a user defined operation to apply to the count

For example, to decrement the 0 counter value by do: `counterAdjust 0 -1`



Caution: currently there is no runtime checking for overflow or underflow; positive-negative wrap-around is typical behavior in such cases.

3. `counterZeroAction` *counterId* { *commandString* }

counterId is in { 0 ... 9 } a user defined identifier for that counter

commandString is an 8-bit char string up to length 255 that defines the action to be formed when the counter value becomes zero following a `counterAdjust` operation.

E.g. `counterZeroAction 0 { stateJump 0 ; midiNote 8 6 2 }`

Continued decrements past zero will produce negative counterValues, but will not trigger the `zeroAction`.

4. `counterModulusAction` *counterId* *modulusValue* { *commandString* }

counterId is in { 0 ... 9 } a user defined identifier for that counter

modulusValue is a positive integer; set to zero to deactivate the modulus check.

commandString is an 8-bit char string up to length 255 that defines the action to be formed when the counter value % `modulusValue` equals zero following a `counterAdjust` operation.

E.g. `counterModulusAction 1 5 { midiNote 8 6 2 }`

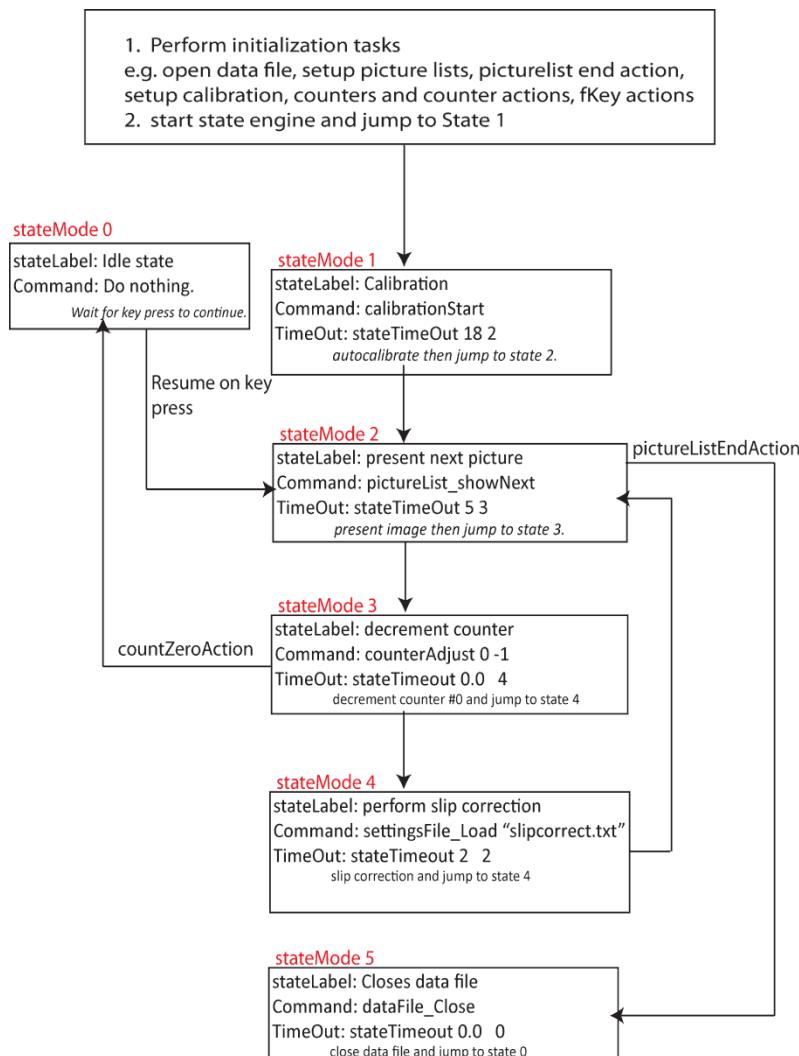
15.4 Miscellaneous

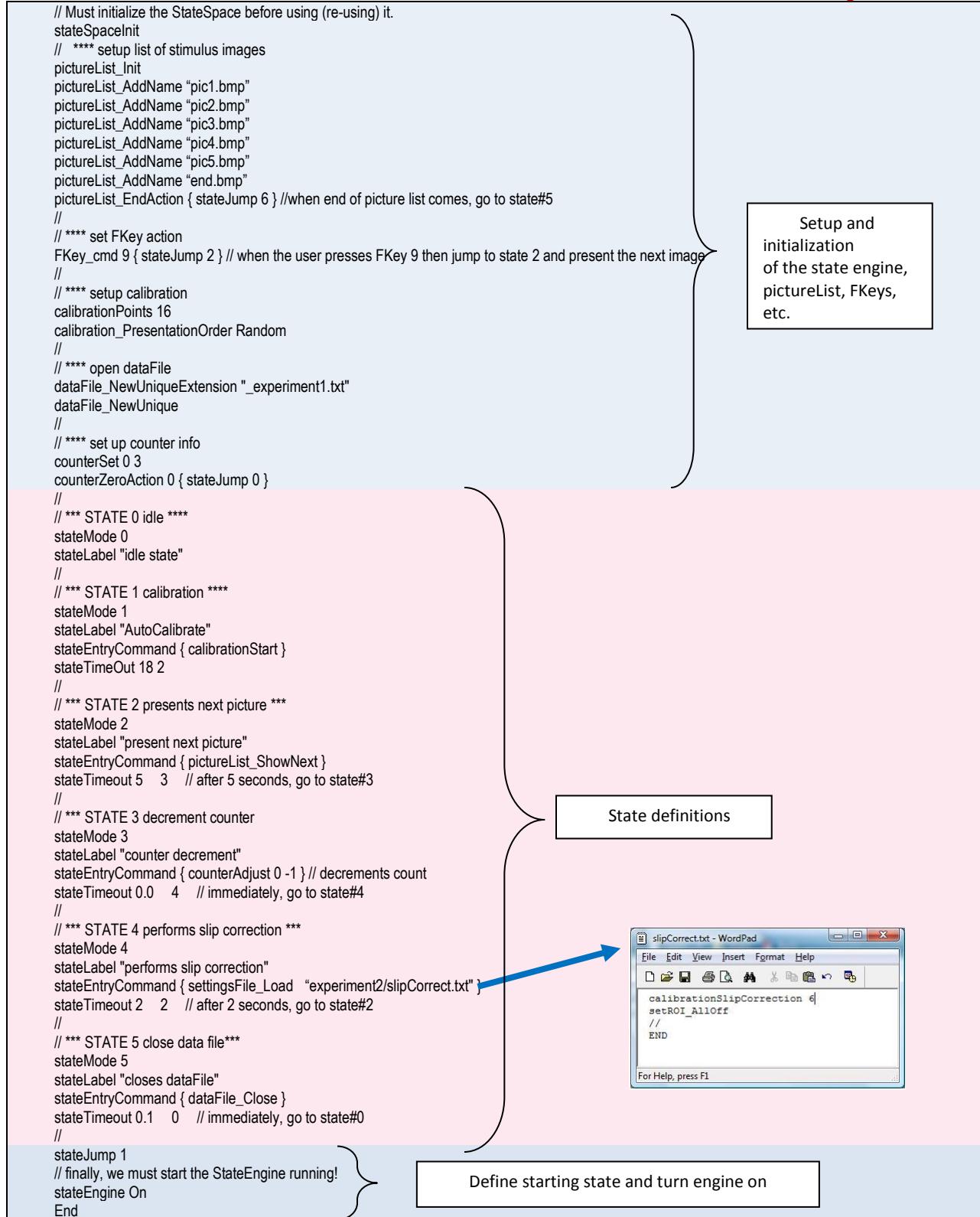
When your experiment incorporates calibration or SlipCorrection, it may be useful to define an action to be performed after they are completed.

Only one state is active at a time. The *StateEngine* controls the transitions between states.

See [Figure 40](#), below for a flow chart of a simple experiment. The user wishes to present five images, for specific times with a slip correction before each image is presented. After 3 images are presented, the image presentation is paused and waits for a key press before continuing.

Figure 40. Simple Experiment Flow example





Chapter 16. Using *Settings* Files

All the Graphical User Interface (GUI) controls in the *ViewPoint* application have equivalent *Command Line Interface* (CLI) strings that are interpreted by the *Command Line Parser* (CLP). All of the GUI values and selections (menu selections, slider values, etc.) can be saved in a *Settings* file that can be later read back into the CLI next time the program is run, so the user can start working without resetting everything by hand. The *Settings* file also contains calibration values, ROI specifications and other program variables. By default, *ViewPoint* assumes that the *Settings* files are stored in the folder:

`~/ViewPoint/Settings/`.

The *Settings* files are in (usually tab delimited) ASCII format. *Settings* files can be saved and loaded using menu selections, as described in [16.2: Saving and Loading Settings Files](#).

The *Settings* file consists of individual lines of ASCII text that may be edited using an editor. The document must however (a) be saved as text only (b) have each command separated by a semicolon, and (c) conform accurately to spelling and spacing requirements. The CLI is not case-sensitive. An editor with good tab setting capabilities is recommended because the row entries are tab separated. The default *Settings* file extension is `.txt` which helps exclude unusable files that contain extra formatting, such as *Rich Text Format* (`.rtf`) and MS Word format files. Use menu item [File > Settings > Edit Settings](#) to select an existing *Settings* file to edit.

Do not modify anything that you do not understand.

Settings Files may be nested (i.e. one *Settings* file may call another *Settings* file, see: [settingsFile_Load](#)). Consequently, it is desirable for each file to be reasonable in length. Currently the *Settings* file is limited to 2000 lines, including comment lines. Note however that the same *Settings* file cannot be called recursively. We encourage you to examine the files in: `~/ViewPoint/Settings/Examples/`.

16.1 CLI String Parsing

All white spaces are gobbled up until the beginning of a string is specified by either (i) a non-white space character, or (ii) a beginning quote is encountered. Quotes should be matched (this was not expected in previous versions). An inline double-forward-slash will cause everything remaining on the line to be ignored as a comment. See [Chapter 18](#).

16.2 Saving and Loading *Settings* Files

[File > Settings > Load Settings](#)

allows the user to read in a *Settings* file, using the standard open file dialog box.

The CLI command: `settingsFolder_Load folderName` loads all the top level (not sub-folders) contents of the specified folder. Note: sub-folder contents may be allowed later, do not assume that this is a safe place for things that you do not want loaded!

[File > Settings > Save Settings](#)

allows the user to store the current settings (except window layout) to a *Settings* file, using the standard open file dialog box.

[File > Settings > Save Window Layout](#)

saves the size, location and z-ordering of all *ViewPoint* windows.

[File > Settings > Verbose Loading](#)

causes additional information from the CLI to be displayed in the *History* window.



16.2.1 *Startup* Folder

When *ViewPoint* starts, it automatically loads all the *Settings* files contained (currently only at the top level) in the folder named: `~/ViewPoint/Settings/Startup/`. This can be used to load regularly used settings to reduce setup time. Small uniquely named files are recommended.

16.2.2 *FinishUp* Folder

When *ViewPoint* stops, it automatically loads all the (currently only top level) *Settings* files contained in the folder named: `~/ViewPoint/Settings/FinishUp/`. This folder is not present in all releases, but can be added by the user.

16.2.3 *Settings/LastRun.txt*

When *ViewPoint* quits it automatically saves all the current control settings in the: `~/ViewPoint/Settings/LastRun.txt` file. The user may load this file manually next time *ViewPoint* is launched. This file is overwritten every time *ViewPoint* quits, so you must rename it if you want it saved. If you want to automatically load this `LastRun.txt` file at program startup, you can add the command `settingsFile_Load "LastRun.txt"` to the file: `~/ViewPoint/Settings/Startup/Startup.txt`

16.3 *Settings File Examples*

It is often a good idea to create individual *Settings* files for different groups of related commands, and then call those *Settings* files from a main *Settings* File.

Example 1: Create individual *Settings* files that contain the name of a bitmap image and the ROIs for that image.

File: `imageAndRoi_1.txt`

```
stimulus_LoadImageFile "picture1.bmp"
setROI_Alloff
setROI_RealRect 1 0.1 0.1 0.3 0.2
setROI_RealRect 2 0.4 0.4 0.5 0.5
```

File: `imageAndRoi_2.txt`

```
stimulus_LoadImageFile "picture2.bmp"
// This has image file has an imageROIfile associated with it.
```

See section [12.8](#)

File: `startup.txt`

```
settingsFile_Load "imageAndRoi_1.txt"
```

Example 2 : Create individual *Settings* files that set the FKey commands for a particular task

File: `FKeysForCalibration.txt`

```
FKey_cmd 9 { calibration_selectPrevious }
FKey_cmd 10 { calibration_snap }
FKey_cmd 11 { calibration_selectNext }
```

16.4 CLI commands

The same *Command Line Interface* (CLI) is used for command strings received from:

- ④ *Settings* files that are loaded.
- ④ The SDK using the `VPX_SendCommand` function
- ④ FKey commands & TTL commands

The total command line length should not exceed 255 characters. See [Chapter 15](#) for details about using CLI s.

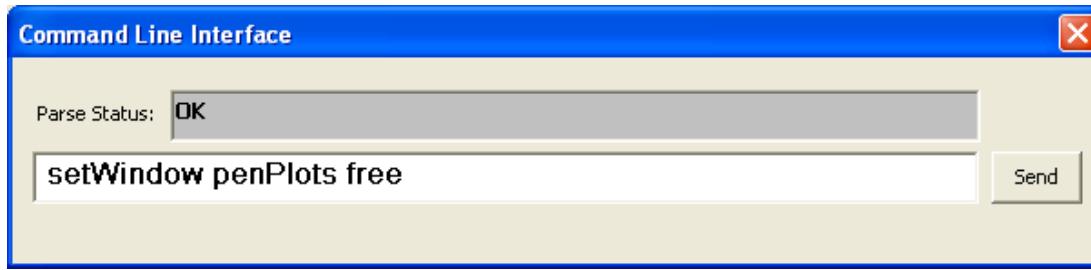
16.5 Associating CLI s with FKeys

CLI s can be associated with FKeys. These associations can be viewed in the Info panel menu: *Help > Info > ShortCuts tab*. Refer to [19.26.1](#)

16.6 Command Line Interface Window

Select menu *Windows > Command Line Interface* to open a simple interface window (see [Figure 41](#)) to send CLI commands to *ViewPoint*. Recently entered commands are saved and can be scrolled using the up/down arrow keys. The interface allows easy cutting and pasting and line editing. The *Command Line Interface* window *Parse Status*: line provides feedback about CLI parsing errors; more detailed error reporting may be found in the *History* window.

Figure 41. Command Line Interface



16.7 Settings File Lists (Deprecated)

A simple sequential state-logic is provided by allowing the user to specify a list of *Settings* files and allowing a variable time delay before loading the next file in the sequence. The *SettingsFileList* to be sequenced through may be set up using a group of CLI (`settingsFileList_Init`, etc.) described in section [SettingsFileList 19.20](#). Start and control the sequencing via the menu item *File > Settings > SettingsFileList*. Hint: It is useful to assign FKey commands for these.



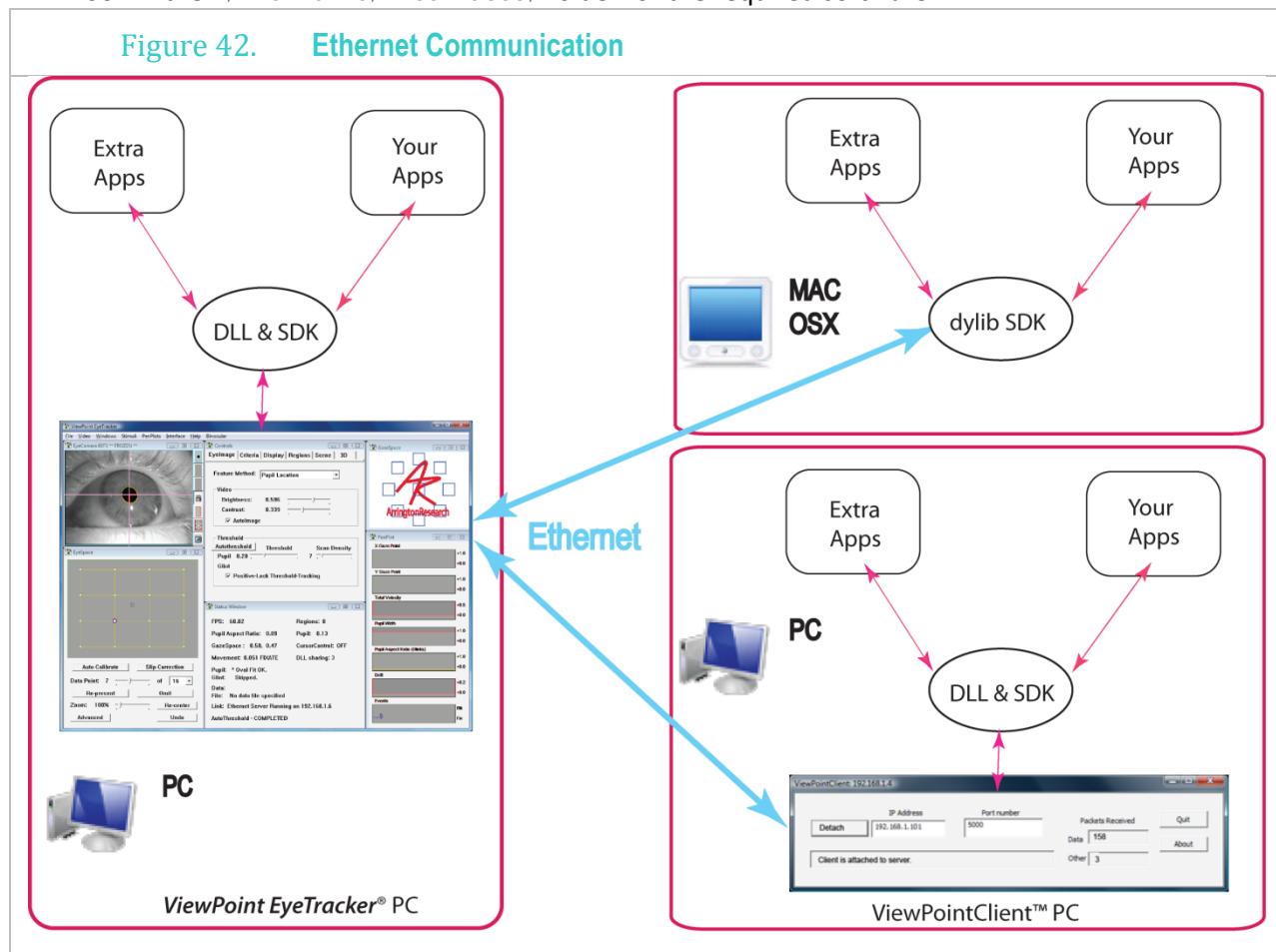
Note that the *SettingsFileList* is deprecated and superseded by the much more powerful *StateEngine*.
See [Chapter 15](#).

Chapter 17. Ethernet Communication Between Computers

The *ViewPoint EyeTracker*® can communicate with other computers via Ethernet. The *ViewPoint EyeTracker* has an Ethernet server built into it. *ViewPointClient™* software can connect to this server and once connected it can send CLI commands to *ViewPoint* and it can receive data from *ViewPoint*.

The interface for “layered” applications running on the remote computer is done via a dynamically linked library. For *Microsoft Windows* this is the familiar `VPX_InterApp.dll`; for *Apple Macintosh* computers this is via `libvpv_interapp.dylib`. The dynamically linked library is part of the *ViewPoint* Software Developers Kit (SDK) that contains high level functions that allow the user to seamlessly and easily interface their programs with *ViewPoint*. The *ViewPointClient* software and the dynamically linked library exchange data with layered applications just like the *ViewPoint EyeTracker* does on the same machine, but it typically takes less than one percent of CPU resources. This means that the same “layered” applications can be used on a remote computer just as easily as on the same computer.

Look in the `~/ViewPoint/Interfaces/` folder for the required software.



The remote computer will of course need an Ethernet connection as well. Users are expected to know how to connect computers via Ethernet. In general this can be done either via an Ethernet cable, or a wireless internet connection to a wireless router on the local network.



You will need to supply the *ViewPointClient* software with the IP address of the computer on the local network that is running *ViewPoint* and also with the port number that *ViewPoint* is using. By default *ViewPoint* uses port 5000. It is rarely necessary to change the port number, only if there is a conflict with another application, so start by leaving it as the default value. We recommend setting a static IP address (one that does not change); to do this, go to the *Windows* control panel and select **Network Connections**. If you need help with setting up the Ethernet connection please contact your local IT support person.

ViewPoint usually shows the correct IP address for the computer on which it is running in the *Status* window and also under menu: *Help > Info > SysInfo tab*.

Note: From the **DOS Command Prompt** use the command **netstat** to access network connection information, including port numbers used. For help, use **netstat – h** to lists all available commands.

17.1 Ethernet Software Connections

Successful Ethernet connection requires that both the IP Address and the Port number must match.

17.1.1 Changing the *ViewPoint* IP Address

Find the IP address of the computer on which *ViewPoint* is running by going to: *Help > Info > SysInfo tab* and look under **NEWORK**: on the **Link:** line. This information may also be available in the *Status* window on the **Link:** line.

We recommend using a static IP address (one that does not change); to do this, go to the *Windows* control panel and select **Network Connections**. If you need help with setting up the Ethernet connection please contact your local IT support person.

Occasionally the system provides the wrong information to *ViewPoint*; if there is a problem check the IP address listed by *Windows* under **Network Connections**.

The current default port is 5000 and should not need to be changed unless there is a conflict.

Extra care must be taken when there is more than one Network Interface Card (NIC) on the *ViewPoint* computer, as problems sometimes arise and the *ViewPoint* software may not list the correct one. Rarely does the server's IP Address need to be changed. However there may be occasions when multiple NIC cards are installed or there is an IP address conflict with other network connections. *ViewPoint* automatically tries to connect with the first NIC listed on the system. If the IP Address is changed, the server will be stopped and all client connections will be lost, then the server will be restarted; any clients will need to be attached again manually. This is done with the following command line instructions CLI:

CLI: ethernet_setIPAddress string
For example: ethernet_setIPAddress "192.168.1.9"

There is no name resolution, so you need to use the IP address not the computer name.

17.1.2 Changing the Port Number

The default Ethernet server port number is 5000 and should not need to be changed unless this port is already in use on your network. If the port is changed, the server will be stopped and all client



connections will be lost, then the server will be restarted; any clients will need to be attached again manually. This is done with the following CLI:

CLI: `ethernet_setPortNumber` *unsignedInteger*
For example: `ethernet_setPortNumber` 5001 -noConfirm

17.1.3 Running the *ViewPoint* Server

It does not matter whether the *ViewPoint* or the *ViewPointClient* application is launched first, however the *ViewPoint* application and its built-in server must be running before the *ViewPointClient* software can attach to the server. Normally the server can remain running and should not need to be turned off, however if this seems necessary, it is accomplished with the following command line instruction:

GUI: Check or uncheck menu: *Interface > Ethernet Server*
CLI: `ethernet_server` *boolValue*
For example: `ethernet_server` OFF

17.1.4 Ping Clients

ViewPoint can send a Ping message to all the attached clients. Each attached client should respond with a Pong message. This can be done with:

GUI: Select menu item: *Interface > Ping Clients*
CLI: `ethernet_PingClients`

When *ViewPoint* receives a Pong response, it calculates the total round-trip time, and prints this time in the *History* window, something like this:

```
42:32:37.929 : : 464) PingPong: 0.217628 milliseconds round trip.
```

This message includes a unique identifier number for each client connection.

17.1.5 Loopback

The server built into the *ViewPoint EyeTracker* and the *ViewPointClient* application can communicate with one another on the same machine (`localhost`) using the *Ethernet* loopback network-interface available at `127.0.0.1`. IPv4 reserves the entire 127 address block (`127.0.0.1` through `127.255.255.254`) for loopback to the same machine, but the name *localhost* usually resolves to address `127.0.0.1`.

If you want to use `127.0.0.1` in the *ViewPointClient*, you must set the IP address in *ViewPoint* to `127.0.0.1` as well, i.e.: `ethernet_setIPAddress` "192.168.1.9". Note that the loopback may not work with some versions of *Microsoft Windows*.

17.1.6 Static IP Addresses and Zero Configuration

If you are connecting your computers together without the help of a DHCP server (a device that assigns IP addresses), *ViewPoint* may start the server with a zero configuration IP address (`169.254.0.0/16`). You should still be able to connect the *ViewPointClient* using this IP Address. However, we have seen some issues where the zero configuration networking doesn't work, so we recommend assigning unique static IP address to both computers.

17.1.7 Firewalls

Many computers now run a firewall to help prevent unauthorized access. You may see a dialog box message similar to the one below if the computer running *ViewPoint* is protected by a firewall program. You must unlock the program, i.e., press the Unblock button, for the *ViewPoint* Ethernet server to communicate with the *ViewPointClient* programs. You may also need to disable any virus checking software.



17.1.8 *ViewPointClient* & *ViewPoint* on the same machine

When using *Ethernet* to communicate with applications on the same computer, the *ViewPoint EyeTracker* server and the *ViewPointClient* software must be in separate folders, each folder with its own copy of the dynamically linked library `VPX_InterApp_nn.dll`.

One reason to use *ViewPointClient* on the same machine is to allow interface to 32-bit applications, as explained next.

17.1.9 *ViewPointClient_32* interface for 32-bit applications

The *ViewPoint* distribution contains 32-bit and 64-bit versions of *ViewPointClient* and the *DLLs*.

`~/ViewPoint/Interfaces/VPx32-Client/` folder contains

`ViewPoint_32.exe` and `VPX_InterApp_32.dll`

`~/ViewPoint/Interfaces/VPx64-Client/` folder contains

`ViewPoint_64.exe` and `VPX_InterApp_64.dll`

The 32-bit *ViewPointClient* and DLL can be used either on a remote machine or on the same machine to interface to 32-bit applications. It can also be used on older 32-bit computers.

For general descriptions we may drop the `_32` and `_64` specification entirely, or use `_nn` instead.

17.2 Ethernet Hardware Connections

17.2.1 Hub, Switch, Router, or a Crossover Cable?

If you already have a Hub, Switch, or Router, then any of these will work fine. If you plan to purchase a new one, then we suggest that you choose a Switch or Router as these are traditionally more efficient. Any of these will allow several computers running *ViewPointClient* to connect to the *ViewPoint EyeTracker* built-in server all at the same time.



If you choose a device that has a DHCP server, like a router, then you must make sure that both computers are connected directly to the same router. You cannot have computer1 connected to router1 while computer2 is connected to router2, this won't work.

17.2.2 Direct Connection using Ethernet Cable.

If you only want to connect two computers then you can simply use an Ethernet Crossover cable, which does not require any kind of Hub, Switch, or Router, but simply plugs into the Ethernet port of each computer. Older computers required a *crossover* cable but most modern computers have NIC cards that resolve the crossover issue and allow you to use a normal Ethernet cable.

17.3 Ethernet to Microsoft Windows computers

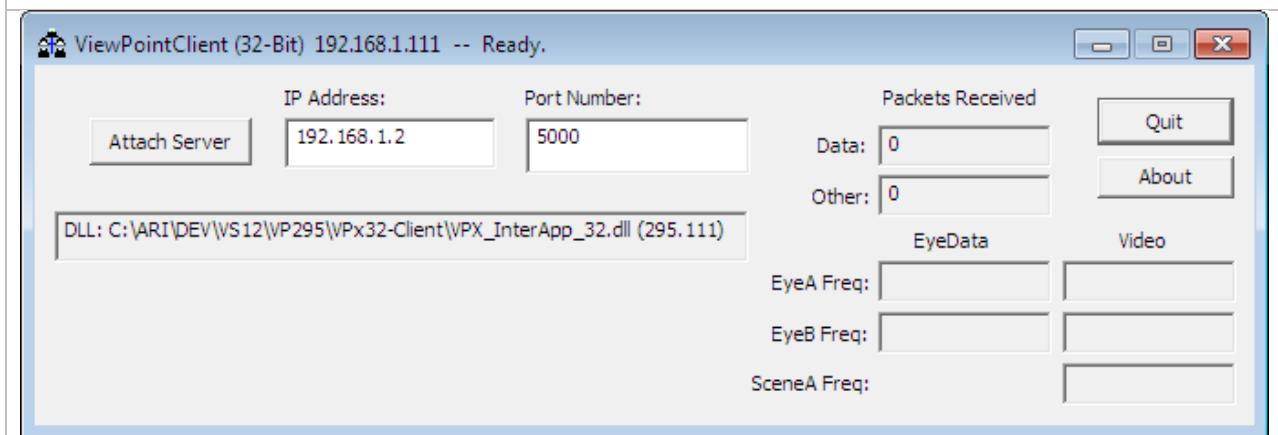
On the remote *Microsoft Windows* computer you will need a copy of the `VPX_InterApp.dll` and a copy of the `ViewPointClient.exe` program. (Note that this is different for the Macintosh where the *ViewPointClient* software is built into the dynamically linked library.) Generally these should be placed in the same folder.

If there are multiple copies of the dynamically linked library on the computer, make sure that the *ViewPointClient* software is connecting to the same copy as the layered application that you are using, otherwise there will be no communication and the data will all be zeroes.

17.3.1 How to use *ViewPointClient*

1. From the `~/ViewPoint/Interfaces/` folder, copy either the `/VPx64-Client/` or `/VPx32-Client/` folder to the second machine. This folder should contain the `ViewPointClient_nn.exe` and the `VPX_InterApp_nn.dll` files. Copy the entire folder contents, as it may contain DLL dependencies.
2. Start the *ViewPoint EyeTracker* application. This automatically starts the built in server.
3. On the remote computer start the `ViewPointClient_xx.exe` application for *Microsoft Windows*; you should see a window like the one in [Figure 44](#).

Figure 44. ViewPointClient Window



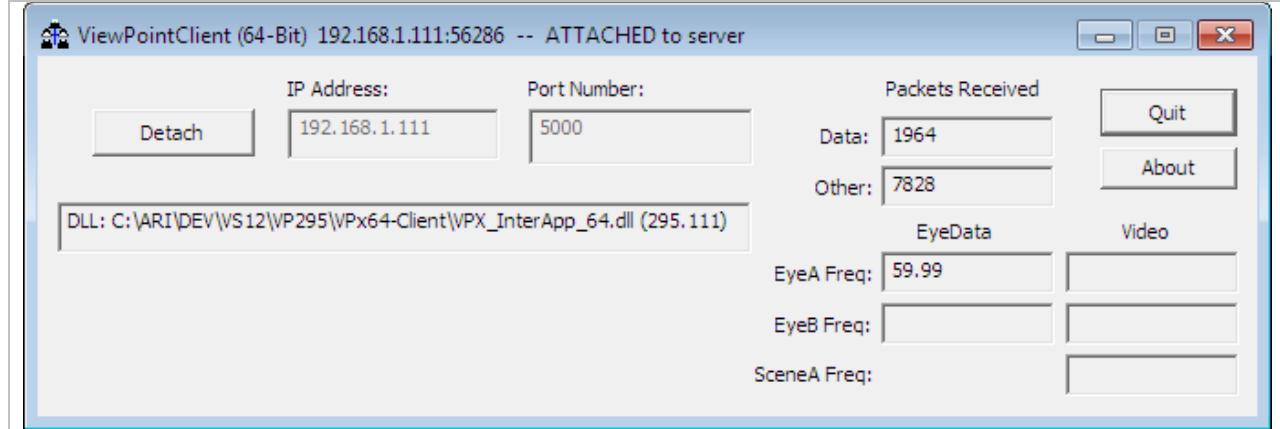
4. Enter the **IP Address** (E.g.: 192.168.1.101) of the computer on which *ViewPoint* is running (see section [17.1.1](#)); do not change the Port number at this time. Click the *Attach Server* button.
5. If connection was successful the button label changes to *Detach*.
6. The changes the *ViewPointClient* application looks something like the following. That shown in [Figure 45](#).

Before attaching to the server, the *ViewPointClient* window will show the full path file name of the DLL that it has loaded. This can be helpful, because layered applications must use the same DLL.

The **EyeData** field indicates the number of eye tracking data packets received. If the **EyeData** field is 0, or remains constant, then probably *ViewPoint* is in *Freeze* video mode. The *ViewPointClient EyeData* field will continuously increment when it is receiving eye tracking data in real-time. The **A Freq** and **B Freq** fields are the calculated frequency from the incoming, streaming packets for each eye. These values should be approximately the selected frame rate in *ViewPoint*.

ViewPoint also sends *other* data even when the video is frozen. Some packets are sent to *ViewPointClient* just to let it know that the connection is still okay, because if the (wireless) connection is lost *ViewPointClient* will automatically attempt to re-attach the connection. Other packets are sent to notify of *ViewPoint* Status changes, ROI events and other events.

Figure 45. ViewPointClient Window when attached



17.3.2 Windows Third Party Applications

The `~/ViewPoint/Interfaces/3rdParty/Microsoft-Windows/` folder contains a number of the third party application interfaces. Documentation for these can be found inside the specific folders. For example, you should find: `~/ViewPoint/Interfaces/3rdParty/Microsoft-Windows/MATLAB/` and `~/ViewPoint/Interfaces/3rdParty/Microsoft-Windows/Python/`.

There are several ways to load the `VPX_InterApp.dll` and use the `ViewPointClient`, below are the suggestions:

Copy the `VPX_InterApp.dll` and the `ViewPointClient.exe` directly into the folder of the third party application being used. This will allow both `ViewPointClient` and the third party application to load the same `VPX_InterApp.dll`. In some versions you may also need to copy the dependency: `opencv_worldnnn.dll` that is in the same folder – be careful: both the 32-bit and 64-bit versions of the OpenCV DLL have the same name; you should be able distinguish them by their date and size.

Some third party applications can load DLL libraries from a specified full path. You should have, both `VPX_InterApp.dll` and the `ViewPointClient.exe` together in their own folder and you can copy the full path to the `VPX_InterApp.dll` that is used by copying it from the text field in the `ViewPointClient` application and then pasting this text string into the third party application's LoadLibrary call specification. This will allow both `ViewPointClient` and the third party application to load the same `VPX_InterApp.dll`.

Another option is to copy the `VPX_InterApp.dll` and the `ViewPointClient.exe` directly into the `System32` folder. You need to make sure this is the only copy of `VPX_InterApp.dll` on this system so that you don't accidentally have the third party application load a different `VPX_InterApp.dll`. The two methods above are preferred, but please note this is a viable option.

17.3.3 Layered Applications

The executable files for the SDK demo-applications are in the

`~/ViewPoint/Interfaces/VPx64-Client/*` folder or in the
`~/ViewPoint/Interfaces/VPx32-Client/*` folder.

You can copy these folders to the second machine. Run the applications just as you would if they were in the folder where `ViewPoint` is running.

These may include:

- `VPX_SimpleC_Callback_Demo_nn.exe`
- `VPX_MFC_Demo_nn.exe`
- `Voltage_nn.exe`

You can also write your own layered applications to run on either machine. We recommend that you start with the sample code and projects in the `ViewPoint/Interfaces/Programming/*` folder.

17.4 Ethernet to Apple Macintosh computers.

The Ethernet interface allows *ViewPoint* running on a *Microsoft Windows* machine to send data and communicate with applications running on a *Macintosh OSX* machine.

Copy `~/ViewPoint/Interfaces/3rdParty/Apple-Mac-OSX/ViewPointMacX-nnn.zip` to your Macintosh computer. Do not unzip the file until it is on the Macintosh! (Some files may be corrupted if you decompressed it on the PC.)

On the Macintosh the *ViewPointClient* software is built into the dynamically linked library (`dylib`), so users no longer need to run an external Client application. There are two new functions in the `dylib` that allow the user to connect to and disconnect from the *ViewPoint EyeTracker* server:

```
int_ VPX_ConnectToViewPoint( char* ipAddress, int32_t port );
    // returns: 0 = connected ok, -1 = generic fail
    // Example: ret = VPX_ConnectToViewPoint("192.168.1.9",5000);

int_ VPX_DisconnectFromViewPoint();
    // returns: 0 = disconnected ok, -1 = generic fail
```

This dylib needs to be copied to a particular folder where applications can know to look. It also needs the permissions set, so that the user can access it. You will need to use the **Terminal** application to do this. The administrator (super user) password will probably be required, because the dynamic library needs to be moved to a low-level part of the UNIX operating system.

1. Launch the *Macintosh Terminal* application from the *Utilities* folder inside the *Applications* folder.
2. Verify that the folder `/usr/local/lib/` exists by listing (`ls`) all the files and folders inside the `/usr/local/` folder; at the command prompt in the Terminal window, enter:

```
ls -l /usr/local/
```

3. You must create the path `/usr/local/lib/` if it does not exist. You will most probably need super-user privileges to do this (`sudo`), which will require the administrator password. To create the `lib/` folder inside the `/usr/local/` folder, first navigate to the `/usr/local/` folder by changing the working directory (`cd`), verify that you got there by printing the path of the working directory (`pwd`), and then make a new director (`mkdir`) folder, finally verify that it exists:

```
sudo cd /usr/local/
pwd
sudo mkdir lib
ls -l
```

4. Copy the library `libvpx_interapp.dylib` into `/usr/local/lib/` so that the layered apps can find it. The tilde stands for the rest of the path name that varies from machine to machine and typically includes the machine name and user name.

```
sudo cp ~/ViewPointMacX/libvpx_interapp.dylib /usr/local/lib/
```

 A quick way to get the full dylib path into the Terminal window rather than typing it in is to simply drag and drop the dylib file into the Terminal window.

5. Navigate to that directory and check for the file

```
sudo cd /usr/local/lib/
```

6. Change the file access permissions:

```
sudo chmod 755 libvpx_interapp.dylib
```

When performing the above steps, it is very useful to check the file attributes before and after executing commands to verify files have been overwritten when copying (i.e. check for newer dates) or that the permissions have been changed when using the `chmod` command. The command to list the file attributes is:

```
ls -l
```

Below is an example of the before and after of using the `chmod` command to change the file permissions. You can clearly see the permissions on the left have changed.

```
-rwx----- 1 root  wheel  285360 Mar 22 19:05 libvpx_interapp.dylib
-rwxr-xr-x  1 root  wheel  285360 Mar 22 19:05 libvpx_interapp.dylib
```

7. Check that everything is working correctly by running the application `CommandLineTool` (in some versions this program is named `ViewPoint ClientTest`). You can just double click the application icon. This launches another simple Terminal window based application; follow the instructions that are printed in the window to set the IP address, connect to *ViewPoint*, send a CLI command, and toggle the printing of streaming data, finally, disconnect from the server; for example:

```
a:192.168.1.79
c
s: say "Hi from the Mac"
i
```

8. Using other applications, connect to *ViewPoint* using the SDK (or application specific wrapper to the SDK). Specify the IP address as a string and the port number as an integer, for example:



```
VPX_ConnectToViewPoint( '192.168.1.99', 5000 );
```

You will need to specify the IP address for the machine on which the *ViewPoint* application is running. You can find the IP address by looking under: *Help > Info > SysInfo tab > Link:* information. Note that the first argument is an ASCII string; some applications will require conversion from Unicode to ASCII, for example *Python* will probably require: `'192.168.1.99'.encode('ascii')`.

9. Before quitting the remote layered program, disconnect the client from the *ViewPoint* server.

```
VPX_DisconnectFromViewPoint();
```

17.4.1 Macintosh Third Party Applications

You will find interfaces to various third party applications inside the `~/ViewPointMacX/` folder that you copied. These include `~/ViewPointMacX/MATLAB/` and in some versions `~/ViewPointMacX/Python/`.

17.4.2 Terminal Window (deprecated method?)

Sometimes it may be necessary to work directly from a Terminal window shell. You can set the dylib path directly in the shell and run applications directly from the shell. Please note this only works for the life of the Terminal window application. Once you close it, you will have to repeat the steps listed below.

- 1) Open a basic Terminal window.
- 2) Set the dylib search path to the *ViewPoint* folder where *libvpx_InterApp.dylib* is located.
Please note that you need to change **ViewPointPath** to your actual path.

```
[export DYLD_LIBRARY_PATH=/ViewPointPath]
```



A quick way to get the full *ViewPoint* path into the *Terminal* window rather than typing it in is to simply drag and drop the *ViewPoint* folder into the *Terminal* window. Please note that you don't want to drag and drop the dylib file because you don't want the file name as part of the path. Make sure you only drag and drop the *ViewPoint* folder.

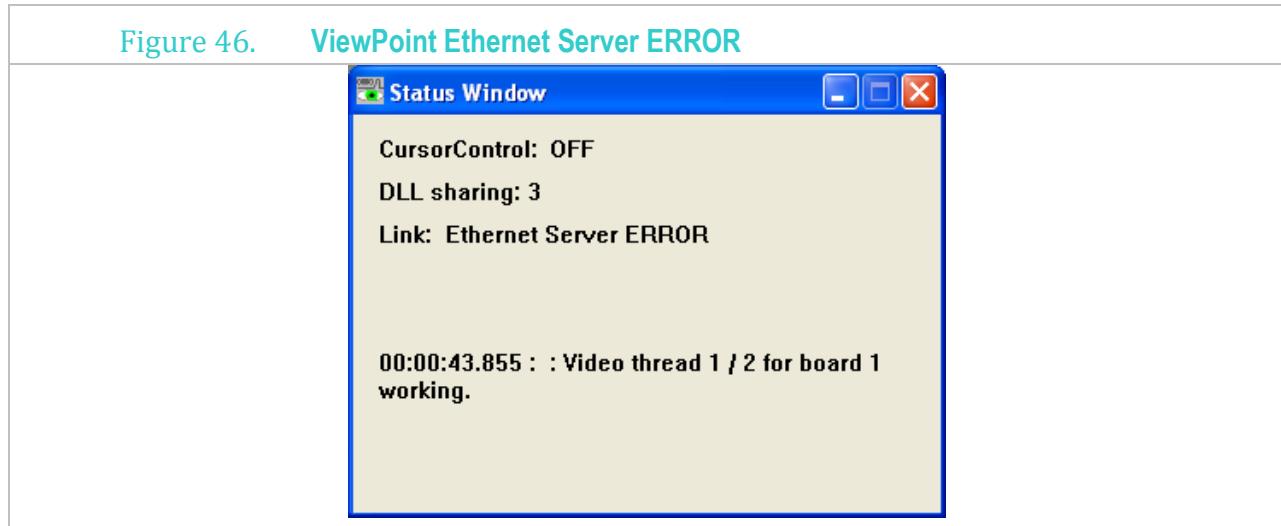
- 3) Run the desired application in the Terminal window, i.e., the *CommandLineTool* application. Simply drag and drop the desired application into the terminal window followed by the return key.

If these steps were not performed correctly, when trying to run the desired application you may get the following error indicating the application could not load the dylib. Please try running the above steps again.

```
dyld: Library not loaded: /usr/local/lib/libvpx_interapp.dylib
Referenced from: /ViewPointMacX/ClientTest
Reason: image not found
Trace/BPT trap
logout
[Process completed]
```

17.5 Ethernet Server Error

When *ViewPoint* starts up or when the server is restarted, there is the potential for the server to have errors when starting, thus leaving it not running. This error can be seen in the *History* window, under: *Help > Info > SysInfo tab* under **NEWORK:** on the *Link:* line. This information may also be available in the *Status* window on the *Link:* line, as shown below in [Figure 46](#).



The most common reasons and their solutions are listed below:

Reason: Two NIC cards installed.

Solution: To verify how many cards are installed use the CLI command: `ethernet_listIPAddresses`. *ViewPoint* prints the NIC info in the *History* window. Verify the appropriate NIC card and change the server's IP Address (see [17.1.1](#)).

Reason: The port number is already in use. If you are on a network, there may be other software using the port or there could be another instance of *ViewPoint* running and that instance already has a server running on that port).

Solution: Change the server's port number (see [17.1.2](#)).

17.6

ViewPoint ClientTest - CommandLineTool

The easiest way to test for Ethernet communication is to use the program **CommandLineTool** (aka **ViewPoint ClientTest**). This lightweight program establishes communication with the *ViewPoint* server, allows sending of CLI commands, and will dump some limited data to demonstrate proper communication. This program works on both the *Microsoft Windows* and the *Apple Macintosh* computers.

```
ViewPoint CommandLineTool
- real-time Ethernet interface to the ViewPoint EyeTracker
```

```
Copyright Arrington Research, Inc.
www.ArringtonResearch.com
DLL version: 293.114014
SDK version: 292.010000
Compiled: 12:18:24 Feb 14 2012
```

Commands:

```
a:192.168.1.4      a:<ipAddressString> no spaces allowed.
p:5000              p:<ipPortNumber> no spaces allowed, unsigned integer.
c                  c connect to ViewPoint EyeTracker.
d                  d disconnect.
i                  i toggles the display of real-time streaming data.
s: say "Hi VP"    s:<cmd> Command Line Interface sent to ViewPoint.
s:client:status    CLI prefix "client:" is caught by the client library.
```

Chapter 18. Command Line Interface (CLI)

The CLI allow users to control almost every aspect of the program and to allow fine control of *ViewPoint* operations and behavior. There is a CLI command for every GUI control. All the CLI commands and their GUI equivalents are listed in [Chapter 19](#), Controls: GUI and CLI.

CLI command **help** will print all the CLI command terms into the *History* window. The list can be narrowed by specifying part of the command term string as an argument, for example: **help video**.

The following is a list of CLI *operators* and their definition. Operators follow a strict precedence which defines the evaluation order of expressions containing these operators.

Table 14. CLI Operators		
Operator	Description	Usage
EOL	End of line (OS specific)	Command terminator
"	double-quote character.	Open and close a <i>string</i> argument.
'	single-quote character.	Open and close a <i>string</i> argument.
//	Double forward-slash	Comment identifier.
{ }	Open and Close braces	Begin and End (deferred) execution block
;	semicolon	Command separator
' ' '\t' ','	Space, Tab, Comma	Argument delimiters
:	colon	target specifier, left-to-right associativity, (no white spaces before the colon)

18.1 Example of a Command Line:

```
EyeB:autothreshold ; FKey_cmd 5 { settingsFile_Load "key 5 .txt" ; dataFile_NewUnique }
```

The above line first starts the *AutoThreshold* process running for EyeB, and then assigns the expression within braces to key **F5**. The expression consists of two instructions that are only evaluated when the **F5** key is pressed, that is, their evaluation is deferred.



Arrington Research

Strings are defined by enclosing a set of characters within quotes ("x"). A string must start with a quote character and end with the same type of a quote character, that is, single or double quotes. An error will be generated if the CLI finds an unmatched quote. Quoted strings have highest precedence, meaning that all text, including operators, inside the quoted string will be shielded from evaluation.

Comments are text that is not evaluated; they are identified by a *comment identifier*, typically two forward slashes (//). A comment identifier tells the CLP that everything following on that line is to be ignored. Comment identifiers can appear anywhere on the line. Note that quotes have precedence, so double slashes inside a quoted string will not be interpreted as comment identifiers.

Braces provide grouping of a sequence of commands, as shown in the above example where the group of instructions is assigned to an FKey. Braces should be used whenever specifying a command that is to have its evaluation deferred. Braces can be nested. An error will be generated if the CLP finds unmatched braces.

The **colon** is used as part of a target specifier. For example, when an instruction could be applied to one or both of eyes, it is used to specify which eye, for example: EyeA:autothreshold. White spaces are not allowed before the colon.

Delimiters separate command arguments. They include white spaces (space-bar and tab characters) and commas.

Semicolons can be used to separate multiple commands on a single line.

18.2 Important CLI Changes from Previous Versions



There are several important and significant changes from previous versions that may require some Settings Files to be modified.

Strings must be delimited by both a beginning and an ending quote. Previously, some commands allowed only a beginning quote.

Only real text strings such as file names should be put in quotes. Previously, deferred command strings were put in quotes.

Deferred commands and command sequences should be put inside matching braces.

Braces may be nested.

Commands separated by semicolons are always evaluated in left-to-right order.

Prefix eye target notation (e.g.: EyeB: autothreshold) is the only specification recognized. Previously, some commands allowed eye target specification as an optional first argument.

18.3 Arguments

Some commands take arguments. Valid arguments include:

Boolean: can be one of the following: yes, no, true, false, on, off, 1, 0, toggle.

Integer: must be numeric digits, can include { + - }; e.g.: -254. Note that the negative sign can also be used to specify the direction of change on TTL lines, so -0 indicates a voltage fall on channel 0 and +0 indicates a voltage rise on channel 0.

Float : must be numeric digits, can include { + - . }; e.g.: 0.75

String : a set of printable ASCII characters inside quotes; e.g.: "This//,that},the :other"

Flags: +option or –option turn various options on and off respectively.

18.4 Asynchronous Operations



Some instructions (e.g. autothreshold, autocalibrate) start asynchronous operations and return before the operation has completed. Incorrect assumptions about sequential execution of instructions may lead to errors that are difficult to debug.

```
int running = VPX_GetStatus( VPX_STATUS_ViewPointIsRunning );
// Wait until true, so parser is running...
int thresh = VPX_GetStatus( VPX_STATUS_AutoThresholdInProgress );
int calib = VPX_GetStatus( VPX_STATUS_CalibrationInProgress );
```

18.5 Error Detection and Reporting

Errors are only detected and reported at the time of evaluation. There is no detection of invalid commands at the time of deferred command assignment; e.g.: when an `FKey_cmd` is assigned.

File name arguments are entered as strings. The validity of file names and folder paths is tested only when the file name is used.

Command arguments should not be in braces unless they are deferred commands. For example:

```
FKey_cmd 5 { settingsFile_Load "my file .txt" } // is valid,
// but penColor { 0 0 255 } // is invalid.
```

18.6 Parameters and Arguments

The terms 'Parameter' and 'Argument' are typically used interchangeably in everyday language. Technically, the *parameters* are the variables used when defining a command, e.g.:

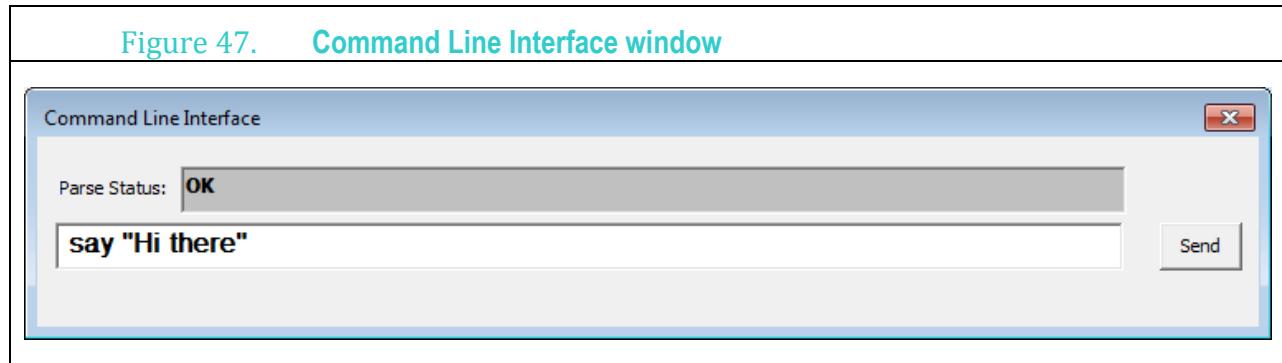
`penColor redVal greenVal blueVal`,
while *arguments* are the specific values passed when calling the command, e.g.: `penColor 0 0 120`.

18.7 Using Commands

There are many ways to issue commands to *ViewPoint EyeTracker*®:

18.7.1 Interactive CLI Window

Select the Menu Item: *Windows > Command Line Interface* to raise a window in which you may issue CLI commands directly to *ViewPoint*.



18.7.2 FKey and TTL

CLI commands may be associated with FKeys for user convenience, and also with TTL inputs (with the TTL option).

18.7.3 Settings Files

Every *Graphical User Interface* (GUI) selection and adjustment that the user makes in *ViewPoint* (e.g., menu item selection, radio button selection, slider value) can be saved in a Settings file, so that they can be loaded again next time the program is run. The control values are stored as single line ASCII commands in the form of a keyword and parameters. When a *Settings* file is loaded, each line in the file is sent to the *ViewPoint Command Line Interface* (CLIParser (CLP)).

18.7.4 SDK/API

These command strings can also be sent to *ViewPoint* from other programs while *ViewPoint* is running, which means that outside, “layered”, programs can have complete control of the *ViewPoint EyeTracker*®. These command strings can be sent via the *Software Developers Kit* (SDK) function `VPX_SendCommand("some command string")`, this can be done from programs running on the same machine or from programs running on remote computers via an Inter-Computer Link.

18.8 Boolean Toggle

All CLI s that accept *BoolValue* arguments (e.g.: *True*, *False*, *On*, *Off*) can also accept the argument *Toggle*, e.g., `dataFile_Pause Toggle`, that changes from the current state to the opposite state.

The power of the **Toggle** argument cannot be overemphasized; it allows a single FKey or TTL pulse to do the work of two, for example, compare the costly way:



```
FKey_cmd 5 { dataFile_Pause } ; FKey_cmd 6 { dataFile_Resume } // takes two keys  
FKey_cmd 7 { videoFreeze ON } ; FKey_cmd 8 { videofreeze OFF } // takes two keys
```

to the efficient way:

```
FKey_cmd 5 { dataFile_Pause Toggle } // one key toggles Pause/Resume  
FKey_cmd 6 { videoFreeze Toggle } // one key toggles Freeze/UnFreeze
```

18.9 Quoting Strings

Literal Strings (e.g., file names, dataFile strings, etc.) can be in either single or double quotes; the only requirement is that the beginning and ending quotes must be the same type to match. This allows strings to be embedded into `VPX_SendCommand` arguments without using \\" (escaped quotes). Previously only double quotes could be used to specify strings to *ViewPoint*.

In *Settings* files or in the Command Line Interface window the following are equivalent:

```
dataFile_InsertString "Say Hello"  
dataFile_InsertString 'Say Hello'
```

C and C++ specify strings with double quotes; in this language the following two commands are equivalent, but the first one is easier to read and less prone to error:

```
VPX_SendCommand( " dataFile_InsertString 'Say Hello' " );  
VPX_SendCommand( " dataFile_InsertString \"Say Hello\" " );
```

MATLAB specifies strings with single quotes, so the following works well:

```
VPX_SendCommand( ' dataFile_InsertString "Say Hello" ' );
```

Python uses the same rules as *ViewPoint*, so the string delimiters can be mixed so long as they are matched.

The *ViewPoint* CLI parser (as of version 2.9.2.11) accepts any of these characters: " (straight double quote, aka Double Prime symbol or inches mark), “ (beginning fancy double quote), and ” (ending fancy double quote), as valid double quotes and any of these characters: ' (straight single quote, aka Prime symbol or feet mark), ‘ (beginning fancy single quote), and ’ (ending fancy single quote), as valid single quotes. This way the user can copy and paste the CLI commands directly from the *ViewPoint UserGuide* without any problems. However, note carefully that the quotes around the `VPX_SendCommand` argument are processed by third party compilers (e.g. *Microsoft VisualStudio*) and interpreters (e.g. MATLAB), many of which will only accept straight quotes (not fancy forward and backward quotes).



Note that the *ViewPoint* CLI parser supports only single byte ASCII characters and single byte character strings; it does NOT support *multibyte characters* such as *Unicode*. Consequently, some compiler and interpreter strings must explicitly be converted to ASCII strings.

For example with some *Python* versions you may need to use the method `str.encode('ascii')` to convert from Unicode to 8-bit ASCII:

```
VPX_SendCommand( str('My Unicode string') ); // works on some Python versions  
VPX_SendCommand( 'My Unicode string'.encode('ascii') ); // more reliable  
VPX_SendCommand( str('My Unicode string').encode('ascii') ); // more reliable  
  
VPX_ConnectToViewPoint( '192.168.1.99'.encode('ascii'), 5000 );
```

18.10 White Spaces

File names cannot begin with a white space. All white spaces are gobbled up until either the beginning of a string is specified by either a non-white space character, or a beginning quote is encountered.

18.11 Case Insensitive CLI Strings

The CLI strings are generally presented starting with lower case and then capitalizing the first letter of successive words, however the parser does not care about the case of the command strings, so you do not need to worry about this as a source of error.

18.12 Embedded Special Characters

The underscore characters in CLI commands are removed before lookup (as of version 2.9.3.115) so for instance, either `datafileNewUnique` or `dataFile_NewUnique` may be used.

18.13 SDK Return Values

All SDK functions return the integer value 1, unless otherwise specified. Check the SDK header file, `VPX.h`, for final authority; changes may appear there before the documentation can be updated.

18.14 VPX_SendCommand & Formatted Strings

The `VPX_SendCommand` function includes a `va_list` mechanism to handle formatted text. This means that it accepts both simple string arguments and also strings with formatting instructions followed by additional arguments, just like the C language `printf` and `scanf` functions. This is extremely convenient for C programming; however, some third party applications (such as MATLAB) do not provide an interface that can handle a `va_list`. To accommodate these we include the `VPX_SendCommandString` function that takes only simple string arguments.

With formatted strings we can simply write:

```
VPX_SendCommand( "stimulus_BackgroundColor %d %d %d", r, g, b );
```

Without formatted strings we must write:

```
char str[255];
sprintf( str, "stimulus_BackgroundColor %d %d %d", r, g, b );
VPX_SendCommandString( str );
```

The programmer should use a *precompiler conditional* found inside the `VPX.h` header file.

```
#ifdef _IMPORTING_INTO_MATLAB
    #define VPX_SendCommand  VPX_SendCommandString
#else
    VPX_DECLSPEC int VPX_SendCommand( char* szFormat, ... );
#endif
```

18.15 TargetPrefix / EyePrefix

Many commands allow specification of the target video stream. For example, in binocular mode you may want to use a command to specify something for one or the other eyes. This is accomplished by using an *EyePrefix*, for example: [EyeB:videoAutoImage ON]. The prefix is separated from the command by a colon; no white spaces are allowed before or after the colon.

Currently, most commands without an *EyePrefix* default to EyeA only, but this is not guaranteed.

Chapter 19. Controls: GUI and CLI

19.1 General

This chapter is the definitive reference for all Graphical User Interface (GUI) and Command Line Interface (CLI) controls. These commands and controls allow users to control every aspect of the program, for example, to adjust the amount of data smoothing, to display / hide various pen plots, freeze / un-freeze the eye camera video. There are commands to open, pause, resume, and close *ViewPoint EyeTracker*® data files.

There is a CLI command for every GUI control; in fact there are many more CLI commands than GUI controls.

The Software Developer's Kit (SDK) includes routines that allow other programs to send the CLI strings as well as providing access to data and other information. The SDK is described in [Chapter 20](#).

19.2 Help finding CLI commands

19.2.1 Help	
GUI:	<i>-none-</i>
CLI :	help help "fileExtensionString"
Default:	
CLI command help will print all the CLI command terms into the <i>History</i> window. The list can be narrowed by specifying part of the command term string as an argument, for example: help video .	
Added: 2.9.4.118	
See also: penPlot_dumpNames	
<pre>help help video VPX_SendCommand("help color");</pre>	

19.3 Data Files

19.3.1 Specify NewUnique Data File Extension

GUI: *-none-*

CLI : **dataFile_NewUniqueExtension "fileExtensionString"**

Default: **.txt**

Specifies the file type extension that is appended when performing the **dataFile_NewUnique** operation. This specification does not affect the format of the data inside the file. This specification does not affect the **dataFile_NewName** command.

A dot should be included for this to correctly specify a file type, e.g. **".wks", ".txt", ".doc"**. The file extension **".wks"**, will usually cause the data file to be opened directly by Microsoft Excel or Microsoft Works spreadsheet packages.

HINT: You can use the extension string to append a group name, e.g., **"_drug_C_.wks"** or **"_MondayData.txt"**.

For help on Quoting Strings see section [18.9](#).

V.2.8.1.12 CLI added.

dataFile_NewUniqueExtension 'drug group X.xls'

```
VPX_SendCommand( "dataFile_NewUniqueExtension '.txt' " );
VPX_SendCommand( "dataFile_NewUniqueExtension 'patient_%d .txt' ", pid );
```



Arrington Research

19.3.2 NewUnique DataFile Name

GUI:	<i>File > Data > Unique Data File ...</i>	<i>^U</i>
	<i>Controls window > Record tab > New Recording button</i>	
CLI :	<code>dataFile_NewUnique</code>	
	<code>dataFile_NewUniqueFormat</code>	
	<code>dataFile_SubstitutionPair</code>	

Opens a new data file with an automatically generated unique file name.

Use `dataFile_NewUniqueExtension` to specify the file extension, e.g.: `".txt"`.

ViewPoint uses a subset of the `strftime` *format specifiers*, e.g. `%Y` is replaced by the current year, `2018`. *ViewPoint* also allows *user defined substitution pairs*, e.g. the user could specify `?E` is to be replaced by another string that can be specified separately. In the example below, `?S` will be replaced by `Karl` , `?E` will be replaced by `0025`, and , `%Y` will be replaced by `2018`.

```
dataFile_SubstitutionPair "?S"      "Karl"
dataFile_SubstitutionPair "?E"      "0025"
dataFile_NewUniqueFormat        "%Y-(Expt=?E)-(Subj=?S)"
dataFile_NewUnique
```

Will produce the following:

`2018-(Expt=0025)-(Subj=Karl)`

The `strftime` specifiers all begin with a percent char, `%`, and are followed by the *specifier char* that must be one of the following: `"aAbBcdHljmpSUwWxXyYz%"`; any other chars are filtered out by replacing the preceding percent char, `%`, with a tilde char, `~`.

The default format string is: `"%Y%m%d_%H%M%S"` .

To reset to the default format, use CLI command:

```
dataFile_NewUniqueFormat DEFAULT
```

The *Controls window > Record* tab shows the data file name and the status of the recording.

See also: `dataFile_NewName`

< <http://www.cplusplus.com/reference/ctime/strftime/?kw=strftime> >

```
VPX_SendCommand( "dataFile_NewUniqueExtension '_patient_%d .txt' ", pid );
VPX_SendCommand( "dataFile_NewUnique" );
```

19.3.3 Open a Data File and Specify a File Name

GUI: *File > Data > New Data File ...* ^N

Controls window > Record tab > New Recording button + Shift-Key

CLI : **dataFile_NewName "fileNameString"**

Opens a new data file with a specified name.

The GUI menu selection allows the user to specify the file name through the *New ViewPoint Data File* popup dialog and allows the user to specify the file name.

The CLI command does NOT popup a dialog; it allows the user to specify the file name. A file extension must be included in the string otherwise none will appear, however an extension may be added later at the operating system level.

The **dataFile_NewUniqueExtension** specification does *not* affect this command.

The *Controls window > Record* tab shows the data file name and the status of the recording.

The file name and path must be inside quotes.

See also: **dataFile_NewUnique**

For help on Quoting Strings see section [18.9](#).

```
VPX_SendCommand( "dataFile_NewName 'myData.txt' " );
VPX_SendCommand( "dataFile_NewName \" C:\VP\Data Files\Exp 6\subj 2.wks\" " );
VPX_SendCommand( "dataFile_NewName \"%s\" ", dataFileName );
```

19.3.4 DataFile AlsoOpen

GUI: **-none-**

CLI : **dataFile_AlsoMake +EyeMovie +Events +History**

ViewPoint allows the user to specify that other files are to be opened and closed at the same time as the *DataFile* and that they will all have the same *newUnique* core name. Precede the other file types with a + to add them or a - to remove them.

```
VPX_SendCommand( " dataFile_AlsoMake -EyeMovie +Events -History " );
```



Arrington Research

19.3.5 Insert a String into the Data File

GUI: *-none-*

CLI : `dataFile_InsertString "UserString"`

Inserts the string into the data file. The string must be inside quotes if it contains white spaces; in C programs the quote characters inside the command string must be escaped with backslashes. The string should be inside quotes.

The string can either be inserted synchronously, i.e., at the end of the next data line (record), or asynchronously, i.e., on a separate line, depending upon the specification of `dataFile_AsynchStringData`. The default is asynchronous.

When requests to insert strings are called more frequently than the data record is saved, and the `dataFile_AsynchStringData` is set to false, then the strings are concatenated. The string separator is the tab character.

For help on Quoting Strings see section [18.9](#).

See also: `dataFile_InsertMarker`, `dataFile_AsynchStringData`

```
VPX_SendCommand("dataFile_InsertString 'showingPictureOfCat' " );
VPX_SendCommand("dataFile_InsertString \" Showing picture of a cat. \\" ");
VPX_SendCommand("dataFile_InsertString \"%s\" ", userString );
```

19.3.6 Insert a Marker into the Data File

GUI: *Windows > Key Pad / Data Marker*

Controls window> Record tab > Mark button

CLI : **dataFile_InsertMarker DataMarker**

DataMarker: Any single byte visible ASCII character.

dataFile_NextMarker

dataFile_RestartMarkers

Insert the specified ASCII character into the data file. This can be used for data synchronization coding, patient responses, etc. The marker character should not be inside any quotes.

The marker can either be inserted synchronously, i.e., at the end of the next data line (record), or asynchronously, i.e., on a separate line, depending upon the specification of **dataFile_AsynchStringData**. The default is synchronous.

The inserted character can be observed in the *Seconds* line graph in the *PenPlot* window.

The *KeyPad / DataMarker* window provides an easy way to manually insert markers into the data file in real-time. NOTE: The *KeyPad / DataMarker* window must have focus (be the active window) for the keyboard keys to be used to insert markers.

Note: In some versions, non-printable ASCII characters are not filtered out, so be careful if you specify such characters as: tab, bell, backspace, line-feed, etc.

dataFile_NextMarker will sequence through the letters 'A' to 'Z'. The sequence restarts after it gets to 'Z'. The sequence can be reset back to 'A' at any time by holding the *Shift-key* when depressing the *Mark button*, or with the CLI command: **dataFile_RestartMarkers**.

See also: **dataFile_InsertString**, **dataFile_AsynchMarkerData**

```
dataFile_InsertMarker K
VPX_SendCommand ("dataFile_InsertMarker K ");
VPX_SendCommand ("dataFile_InsertMarker %c ", theMarker );
```



19.3.7 Insert a User Defined Tag into the Data File

GUI: *-none-*

CLI : **dataFile_InsertUserTag UserTagNumber "UserString"**

UserTagNumber: an integer in the range 800 to 899

This allows users to insert data from their own sources into a data file with their own specified tag. The tag appears in column 1 of the data file. The insertion is done asynchronously with respect to the eye movement data records and the insertions are uniquely time stamped. Tag identifiers must be in the range 800-899.

See: Section [14.3.5](#) on page 86. for more details.

For help on Quoting Strings see section [18.9](#).

See also: [dataFile_InsertString](#)

dataFile_InsertUserTag 800 "MyUserInfo 1 A 0.888"

VPX_SendCommand("dataFile_InsertUserTag 800 'MyInfo\t%c\t%s\t%d' ", ch, szT, nVal);

19.3.8 Specifies Asynchronous or Synchronous String Data

GUI: *File > Data > Asynchronous String Data*

CLI : **dataFile_AsynchStringData BoolValue**

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Yes

Specifies whether to insert string data asynchronously or synchronously into the data file.

Synchronously means that this data is appended to the same line as the normal eye tracker data.

This string data will be treated as multi-column data by using tab-characters as column separators.

Synchronous data is usually easier to load into spread sheets or other analysis packages. If several Strings are inserted between eye tracker samples, the Strings will all be concatenated. If this control is set to Yes (i.e., Asynchronous), then each String is separately time stamped and inserted on a data line by itself.

Section [14.3.3](#) describes synchronous vs. asynchronous operations.

See also: [dataFile_InsertString](#)

VPX_SendCommand("datafile_AsynchStringData No");



Arrington Research

19.3.9 Specify Asynchronous or Synchronous Marker Data

GUI: *File > Data > Asynchronous Marker Data*

CLI : **dataFile_AsynchMarkerData BoolValue**

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default Setting: No

Specifies whether to insert data markers asynchronously or synchronously into the data file.

Synchronously means that this data is on the same line as the normal eye tracker data, in a separate column, which is usually easier to load into spread sheets or other analysis packages. If several Markers are inserted between eye tracker samples, the Markers will all be displayed together and more precise Marker time information is lost. If this control is set to Yes (i.e., Asynchronous), then each Marker event is separately time stamped and inserted on a data line by itself.

Section [14.3.3](#) describes synchronous vs. asynchronous operations.

See also: [dataFile_InsertMarker](#)

```
VPX_SendCommand( "datafile_AsynchMarkerData Yes" );
```

19.3.10 Specify Asynchronous or Synchronous Head Tracker Data

GUI: *File > Data > Asynchronous Head Tracker Data*

CLI : **dataFile_AsynchHeadData BoolValue**

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Yes

Available only with head tracker option

Specifies whether to insert head tracker data asynchronously or synchronously into the data file.

Section [14.3.3](#) describes synchronous vs. asynchronous operations.

```
VPX_SendCommand( "dataFile_AsynchHeadData yes" );
```



19.3.11 Specify Data File Start Time

GUI: *File > Data > Start Data File Time At Zero*

CLI : *dataFile_startFileTimeAtZero BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default Yes
Setting:

Specifies whether to start each new data file at time=0. Otherwise, the data-file will use the time from when the DLL was initialized, so the time values in each sequential data file will be increasing and represent actual elapsed time. The DLL time starts at zero when the DLL is launched, which is when the first program that accesses it is launched. Turning this off may be useful to keep track of fatigue factors or the duration of rest periods. It may also be useful because other programs that use the DLL time will have the same time values, which can aid in post-hoc synchronization of events.

Note: version 2.9.3.120 introduces a third SDK option, SINCE_SYSTEM_INIT_TIME, but it is not currently implemented for the *DataFile*.

See also: [VPX_GetPrecisionDeltaTime](#)

`VPX_SendCommand("datafile_StartFileTimeAtzero No");`

19.3.12 Include Raw Eye Data in Data File

GUI: *File > Data > Include Raw (unmapped EyeSpace) Data*

CLI : *dataFile_includeRawData BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: No

Specifies whether to store raw eye data in the data file.

`VPX_SendCommand("dataFile_includeRawData Yes");`

19.3.13 Include Events Data File

GUI: *File > Data > Include Events in Data File*

CLI : *dataFile_includeEvents BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: No

Specifies whether to store contents of the Events window into the data file.

`VPX_SendCommand("dataFile_includeEvents Yes");`



19.3.14 Specify Whether to Use *DataFile* Buffering (DEPRECATED)

GUI: *-removed-*

CLI : *dataFile_UseBuffering BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Yes

DEPRECATED

Allows selection between (Yes) buffers the data in RAM before saving to disk or (NO) immediately write to disk each data record. Because previous versions *Microsoft Windows* were quite unstable, many users preferred to turn buffering off, so that data was not lost in the event of a system or program crash; this did however sometimes incur a slower sampling rate. In general this is neither required nor recommended with most new operating systems.

`VPX_SendCommand("dataFile_UseBuffering No");`

19.3.15 Pause Writing of Data to File

GUI: *File > Data > Pause Data Capture (toggle)* *^P*

Controls window > Record tab > Pause/Resume button

CLI : *dataFile_Pause BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

dataFile_Resume // No arguments.

Pauses the writing of data to an open data file. Inserts a “+” marker into the data file when paused and inserts a “=” marker at time resumed. Because of the *Microsoft Windows* overhead for opening and closing files, the user may prefer pausing and resuming to opening and closing. Also, pause may be set before the file is opened, such that the overhead delays for opening the file are finished before the start of the experiment.

The *Controls window > Record* tab shows the data file paused state.

The CLI command *dataFile_Pause* accepts a Boolean argument allowing for the CLI instruction *dataFile_Pause Toggle* to toggle pause on and off with a single instruction.

See also: *videoFreeze*, *dataFile_UnpauseUponClose*, *videoFreezeSync*

`FKey_cmd 9 { dataFile_Pause Toggle }`
`VPX_SendCommand("dataFile_Pause OFF");`
`VPX_SendCommand("dataFile_Resume");`

19.3.16 Opening Data File in Paused State

GUI: *-none-*

CLI : *dataFile_UnpauseUponClose BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Yes (as of version 2.8.4.569)

If this is ON, then closing a *DataFile* removes the paused state; if the user wants the next *dataFile* to start paused, the *Pause* button will need to be pressed again.

```
VPX_SendCommand( "dataFile_UnPauseUponClose False" );
```

19.3.17 Close Data File

GUI: *File > Data > Close Data File* ^W

Controls window > Record tab > Close button

CLI : *dataFile_Close*

Closes a data file if one is open, regardless of the Paused state.

The *Controls window > Record tab* shows the current data file state.

See also: *dataFile_CloseAndAnalyze*

```
VPX_SendCommand( "dataFile_Close" );
```

19.3.18 Close Data File and Open in Post-Hoc Analysis tool

GUI: *File > Data > Close and Analyze Data File* Alt-Shift-W

Controls window > Record tab > Analyse button

CLI : *dataFile_CloseAndAnalyze*

Closes a data file if one is open, regardless of the Paused state, and automatically launches the specified data analysis program with the last data file loaded in. The default data analysis program is: “~/ViewPoint/DataAnalysis.exe”, but this can be changed using CLI:

dataAnalysisApp

The *Controls window > Record tab* shows the current data file state.

```
VPX_SendCommand( "dataFile_CloseAndAnalyze" );
```



19.3.19 DataAnalysis Application

GUI: *-none-*

CLI : `dataAnalysisApp "fileNameString"`

The default data analysis program is: `"~/ViewPoint/DataAnalysis.exe"`, but this can be changed using CLI: `dataAnalysisApp`

```
VPX_SendCommand("dataAnalysisApp 'C:/ARI/anotherAnalysis.exe' ");
VPX_SendCommand("dataFile_CloseAndAnalyze");
```

19.4 Corrected Data

19.4.1 Geometry Window

GUI: *Stimuli > Geometry Setup ...* *Ctrl-G*

CLI : `setWindow Geometry Show`

Raises the *Geometry* window.

The tabs in this widow can be selected using the following:

`geometryTab 2D; geometryTab parallax; geometryTab pupilScale`

```
VPX_SendCommand("setWindow Geometry Show; geometryTab pupilScale" );
```

19.4.2 Geometry Measurement

GUI: *Geometry window > 2D tab > Sliders*

CLI : `geoViewDistance IntValue`

`geoVerticalMeasure IntValue`

`geoHorizontalMeasure IntValue`

These CLI commands allow the user to set the measured values for the stimulus geometry calculations.

Generally these are considered to be in millimeters, but they could be in any unit so long as they are all the same units.

```
VPX_SendCommand( "geoViewDistance 50" );
VPX_SendCommand( "geoVerticalMeasure 500" );
VPX_SendCommand( "geoHorizontalMeasure 400" );
```



Arrington Research

19.4.3 Geometry Grid Spacing

GUI: *-none-*

CLI : *gridSpacing floatMinor floatMajor*
 floatMinor, floatMajor: floating point numbers

Specifies the grid spacing to be viewed in the *GeometryGrid*.

```
gridSpacing 2.5 10  
// sets the minor axes every 2.5 degrees, major axes every 10 degrees  
VPX_SendCommand("gridSpacing 2.5 10" );
```

19.4.4 GeometryGrid Lines Display

GUI: *Controls window > Display tab > GeometryGrid check boxes*

CLI : *GazeSpaceGraphicsOptions +Grid*
 StimulusGraphicsOptions +Grid
 SceneMovieGraphicsOptions +Grid

Specifies the display of the *GeometryGrid* on the *GazeSpace* window, *Stimulus* window, or the *SceneMovie* when available. Use minus sign to remove, e.g.: *-Grid*

```
VPX_SendCommand("SceneMovieGraphicsOptions +Grid" );
```

19.4.5 Specify Amount of Parallax Correction

GUI: *Geometry window > Parallax tab > Slope slider*

CLI : *parallaxCorrection_Slope float*
 in range -2.0 to 2.0

Default: 0.82

This command sets the amount of parallax correction. Only applicable to the Binocular *SceneCamera* systems. Refer to [9.2](#)

```
VPX_SendCommand ( "parallaxCorrection_Slope 0.8" );
```



Arrington Research

19.4.6 Inter-Pupillary Distance (IPD) Measure

GUI: *-none-*

CLI : ipdMeasure *millimeters* in range 1.0 to 99.0

Default: 60

Accurate calcuation of **3D GazePoint** and **3D Vergence** calculations require accurate **ipdMeasure**, **geoHorizontalMeasure**, **geoVerticalMeasure** and **geoViewingDistance** measurements be set.

Note that **ViewPoint** assumes that the vertical eyeball position is at the vertical center of the **Stimulus** or **SceneCamera** window. For more flexibility, use **3DViewPoint** or **3DWorkSpace** products.

Adult Human Male USA: 55mm – 70mm (5th-95th percentile)

Adult Human Female USA: 53mm – 65mm (5th-95th percentile)

Human Child USA: 41mm – 55mm (low to high)

2.9.3.121 argument changed from integer to float.

```
VPX_SendCommand ( "ipdMeasure 59.5" );
```

19.4.7 Pupil Diameter Calibration

GUI: *Geometry window > PupilScale tab > [Set EyeA] and [Set EyeB] buttons*

CLI : EyeA:pupilScaleFactor *float*

EyeB:pupilScaleFactor *float*

in range -2.0 to 2.0

Default: 0.82

The **Pupil Scale Factor** is used that convert normalized units to millimeters and the result will not be correct unless this calibration is performed correctly in advance.

See: [VPX_GetPupilDiameter2](#)

```
VPX_SendCommand ( "parallaxCorrection_Slope 0.8" );
```

19.5 Stimulus Images

19.5.1 Load Stimulus Image into the Stimulus window

GUI:	<i>File > Images > Load Image</i>	▲
CLI :	<code>stimulus_LoadImageFile fileName</code>	
Loads the selected image into the <i>Stimulus</i> window. For help on Quoting Strings see section: 18.9 . See also: <code>stimulusGraphicsOptions +Image</code>		
<pre>VPX_SendCommand("stimulus_LoadImageFile catPicture.bmp"); VPX_SendCommand("stimulus_LoadImageFile \"second cat picture .bmp\" "); VPX_SendCommand("stimulus_LoadImageFile %s", bitmapFileName);</pre>		

19.5.2 Specifies How to Display the Currently Loaded Stimulus Image

GUI	<i>Stimuli > Image Shape > shape type</i>
CLI :	<code>stimulus_ImageShape ShapeType</code> <i>ShapeType: Actual, Centered, Fit, Isotropic</i>
Default:	Fit
Specifies how the bitmap image is to be displayed in the <i>Stimulus</i> and <i>GazeSpace</i> windows.	
Actual: Displays the image at actual size with the top-left corner of the image positioned at the top-left corner of the display window. Centered: Displays the image at actual size and centered in the window. Fit: Displays the image stretched un-equally to fit the window. Isotropic: Displays the image stretched equally in all directions, so as to maintain the original proportions, i.e., the original aspect ratio of the image. This may leave window background ("matting") color between edges of the picture and the edges of the window. The color of this area can be specified with the command: <code>stimulus_BackgroundColor</code> . Note: the <i>GazeSpace</i> window may automatically resize to match the image and matting shown in the <i>Stimulus</i> window. See also: <code>stimulus_BackgroundColor</code> <code>VPX_STATUS_StimulusImageShape</code>	
<code>VPX_SendCommand("stimulus_ImageShape Isotropic");</code>	



Arrington Research

19.5.3 Specify a Background "Matting" Color for the Stimulus Window

GUI: *Stimuli > Background Color*

CLI : `stimulus_BackgroundColor intRed intGreen intBlue`
`intRed 0-255 intGreen 0-255 intBlue 0-255`

Sets the background ("matting") color for use when *ImageShape* is not set to *Fit* and there is space at the sides or at the bottom of the image.

See also: `stimulus_ImageShape`

`stimulus_BackgroundColor 255 0 0 // set to BRIGHT RED`

`VPX_SendCommand ("stimulus_BackgroundColor 255 135 75");`

19.5.4 Stereo Display Mode (Side-by-Side)

GUI: *Stimuli > Stimulus Window Properties > Stereo Display (toggle)*

CLI : `stereoDisplay BoolValue`
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

For displaying a single stimulus image on a 3D display device that expects a side-by-side stereo image format.

Two sets of calibration StimulusPoints and ROIs are displayed, one for each eye, in each of the side-by-side frames. These can be different, in the case of *Partial Binocular Overlap*, see [8.12.4.2](#).

New: 2.8.3.40, behavior modified 2.9.5.142

See also: `stereoPseudo` `stereoSwapEyes`

`VPX_SendCommand ("stereoDisplay YES");`

19.5.5 Create a Fake, or Pseudo, Stereo Image (Side-by-Side)

GUI: *-none-*

CLI : `stereoPseudo BoolValue`
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Two identical side-by-side images are created from the current loaded stimulus image for display in the *Stimulus* window and in the *GazeSpace* window, which can be used as a 3D stereo pair.

Requires: `stereoDisplay ON`

New: 2.9.5.142

See also: `stereoDisplay` `stereoSwapEyes`

`VPX_SendCommand ("stereoPseudo YES");`

19.5.6 Swap EyeA and EyeB for Stereo Image

GUI: Stimuli > Stimulus Window Properties > Stereo Swap Eyes (toggle)

CLI : `stereoSwapEyes BoolValue`

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

When displaying 3D, stereo images, this changes the assumption that EyeA is on the Left.

Requires: `stereoDisplay ON`

New: 2.9.5.142, replaces deprecated misnomer `stereoSwapImages` that did the same thing.

See also: `stereoDisplay` `stereoPseudo`

```
VPX_SendCommand ( "stereoSwapEyes YES" );
```

19.5.7 Play specified Sound File

GUI: `-none-`

CLI : `stimulus_PlaySoundFile soundFileName`

Plays the specified sound file. May be used as an auditory cue. If the string contains spaces it must be in quotes. e.g. `stimulus_PlaySoundFile "Yes.wav"`.

For help on Quoting Strings see Section: [18.9](#)

This feature may cause a media/audio player to open, if this happens, check your computer settings. If this problem persists, try using midi notes.

```
stimulus_PlaySoundFile "a very loud meow .wav"
VPX_SendCommand( "stimulus_PlaySoundFile 'meow.wav' " );
VPX_SendCommand( "stimulus_PlaySoundFile \"a very loud meow .wav\" " );
VPX_SendCommand( "stimulus_PlaySoundFile \"%s\" ", soundFileName );
```

19.6 PictureList

See [12.3](#) for a description of *PictureLists*.

19.6.1 Initialize Picture List

GUI: *-none-*

CLI : *pictureList_Init*

Initializes the list for stimulus images, making it ready for new names to be entered.

```
VPX_SendCommand( "pictureList_Init" );
```

19.6.2 Add New Image Name to Picture List

GUI: *-none-*

CLI : *pictureList_AddName imageFileName*

Adds an image file name to the picture list.

```
pictureList_AddName "picture of cat.bmp"
```

```
VPX_SendCommand ( "pictureList_AddName 'picture of cat .bmp' " );
```

```
VPX_SendCommand ( "pictureList_AddName \\\"%s\\\" ", imageFileName );
```

19.6.3 Randomize List of Images in the Picture List

GUI: *File > Images > Picture List > Randomize PictureList*

CLI : *pictureList_Randomize*

Randomizes the pointers in the picture list. Repeat this to re-randomize.

```
VPX_SendCommand( "pictureList_Randomize" );
```

19.6.4 Move to Next Image in the PictureList

GUI: *File > Images > PictureList > Next PictureList Image ^F12 (default)*

CLI : *pictureList_ShowNext*

Moves to the next file pointer in the picture list.

```
VPX_SendCommand ( "pictureList_ShowNext" );
```



Arrington Research

19.6.5 Move to Start of Images in Picture List

GUI: *File > Images > PictureList > Restart PictureList*

CLI : **pictureList_Restart**

Re-sets the pointer to the first image in the list. This does not un-randomize or re-randomize if the list has been randomized.

```
VPX_SendCommand ( "pictureList_Restart" );
```

19.6.6 Picture List End Action

GUI: *-none-*

CLI : **pictureList_EndAction**

Specifies a command to be executed when the end of the picture list is reached.

```
VPX_SendCommand ( "pictureList_EndAction {stateJump 5}" );
```

19.7 Controls Window: EyeImage

The tabs in the *Controls* window can be selected using the following:

controlsTab EyeA

controlsTab EyeB

controlsTab Scene

controlsTab Criteria

controlsTab Display

controlsTab Regions

controlsTab Record



19.7.1 Specify Mapping Feature

GUI: *Controls Window > EyeA or EyeB tab > Feature Method pull down list*

CLI : *mappingFeature Method //sets Eye A*

EyeA:mappingFeature Method

EyeB:mappingFeature Method

Methods: Pupil, Glint, Vector, SlipComp

Default: **Pupil**

Controls what type of mapping will be done from *EyeSpace* to *GazeSpace* for the specified eye.

Slip Compensation provides a means to compensate for HMD slippage.

```
VPX_SendCommand ( "mappingFeature pupil" ); // Eye A
```

```
VPX_SendCommand ( "EyeA:mappingFeature vector" );
```

```
VPX_SendCommand ( "EyeB:mappingFeature vector" );
```

19.7.2 AutoThreshold

GUI: *Controls window > EyeA or EyeB tab > Threshold Group > Autothreshold button*

CLI : *autoThreshold // Eye A*

EyeA:autoThreshold

EyeB:autoThreshold

Tries to automatically set desirable glint and pupil threshold levels for the specified eye based on the current eye image. For continuous threshold adjustment use **positiveLock**.

```
VPX_SendCommand ( "autoThreshold" ); // Eye A
```

```
VPX_SendCommand ( "EyeA:autoThreshold" );
```

```
VPX_SendCommand ( "EyeB:autoThreshold" );
```

19.7.3 Positive Lock Tracking

GUI: *Controls window > EyeA or EyeB tab > Threshold group > Positive-Lock Threshold-Tracking checkbox*

CLI : *positiveLock BoolValue //sets Eye A*

EyeA:positiveLock BoolValue

EyeB:positiveLock BoolValue

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: **On**

Continuous automatic feature threshold adjustment for the specified eye.

```
VPX_SendCommand ( "positiveLock ON" ); // Eye A
```

```
VPX_SendCommand ( "EyeB:positiveLock ON" );
```



Arrington Research

19.7.4 Adjust Pupil Threshold Slider

GUI: *Controls window > EyeA or EyeB tab > Threshold Group > Pupil Threshold slider*

CLI : *pupilThreshold NormalizedValue // Eye A*

EyeA:pupilThreshold NormalizedValue

EyeB:pupilThreshold NormalizedValue

NormalizedValue: a floating point number in range 0.0 to 1.0

Default: 0.25, but **autoThreshold** at startup may change this

Sets the image intensity threshold for the specified eye such that the pupil can be segmented from the rest of the image. The assumption is that the pupil is darker than the rest of the image within the PupilScanArea. For binocular systems, the user may specify this value separately for each eye.

The value zero represents black, the lowest pixel intensity possible, and the value one represents white, the highest pixel intensity possible.

See also: **autoThreshold**

```
VPX_SendCommand ( "pupilThreshold 0.7" ); // eye A  
VPX_SendCommand ( "EyeA:pupilThreshold 0.7" );  
VPX_SendCommand ( "EyeB:pupilThreshold 0.7" );
```

-



19.7.5 Adjust Glint Threshold Slider

GUI: *Controls window > EyeA or EyeB tab > Threshold Group > Glint Threshold slider*

CLI : *glintThreshold NormalizedValue // Eye A*

EyeA:glintThreshold NormalizedValue

EyeB:glintThreshold NormalizedValue

NormalizedValue: a floating point number in range 0.0 to 1.0

Default: 0.88, but AutoThreshold at startup may reset this

Sets the image intensity threshold for the specified eye such that the glint (aka, corneal reflection, corneal reflex, or 1st Purkinje image) can be segmented from the rest of the image. The assumption is that the glint is brighter than the rest of the image within the GlintScanArea. For binocular systems, the user may specify this value separately for each eye.

The value zero represents black, the lowest pixel intensity possible, and the value one represents white, the highest pixel intensity possible.

See also: [autoThreshold](#)

```
VPX_SendCommand ( "glintThreshold 0.6" ); // Eye A
```

```
VPX_SendCommand ( "EyeA:glintThreshold 0.6" );
```

```
VPX_SendCommand ( "EyeB:glintThreshold 0.6" );
```

19.7.6 Adjust Video Image Brightness

GUI: *Controls window > EyeA or EyeB tab > Video Group > Brightness slider*

CLI : *videoImageBrightness NormalizedValue // sets EyeA*

EyeA:videoImageBrightness NormalizedValue

EyeB:videoImageBrightness NormalizedValue

SceneA:videoImageBrightness NormalizedValue

NormalizedValue: a floating point number in range 0.0 to 1.0

Default: - varies -

Sets the video image brightness level of the specified eye, normalized from 0.0 to 1.0.

SceneA: prefix added in version 2.9.5.143

```
VPX_SendCommand( "videoImageBrightness 0.5" ); //sets EyeA
```

```
VPX_SendCommand( "EyeA:videoImageBrightness 0.5" );
```

```
VPX_SendCommand( "EyeB:videoImageBrightness 0.5" );
```



Arrington Research

19.7.7 Adjust Video Image Contrast

GUI:	<i>Controls window > EyeA or EyeB tab > Video Group > Contrast slider</i>
CLI :	<code>videoImageContrast NormalizedValue // sets EyeA</code> <code>EyeA:videoImageContrast NormalizedValue</code> <code>EyeB:videoImageContrast NormalizedValue</code> <code>SceneA:videoImageContrast NormalizedValue</code> <code>NormalizedValue: a floating point number in range 0.0 to 1.0</code>
Default:	- varies -
	Sets the video image contrast level of the specified eye, normalized from 0.0 to 1.0. SceneA: prefix added in version 2.9.5.143
	<code>VPX_SendCommand("videoImageContrast 0.7"); //sets Eye A</code> <code>VPX_SendCommand("EyeA:videoImageContrast 0.7");</code> <code>VPX_SendCommand("EyeB:videoImageContrast 0.7");</code>

19.7.8 Camera Shutter / ExposureTime

GUI:	<i>Controls window > EyeA or EyeB tab > Video Group > AutoImage checkbox</i>
CLI :	<code>OptionalPrefix:videoShutter exposureTimeMilliseconds</code> <code>OptionalPrefix:videoShutter argument</code> <code>exposureTimeMilliseconds: a floating point number greater than zero</code> <code>argument: one of { Auto, Report }</code> <code>OptionalPrefix: one of { EyeA: EyeB: SceneA: } with colon and no separating whitespace.</code> <code>Omitting the prefix defaults to EyeA.</code>
Default:	On
	Only for uEye digital cameras. Specifying a real number sets the exposure time to that value in milliseconds. <code>Report</code> will print the current value in the <i>History</i> window. <code>Auto</code> will instruct the camera to automatically adjust the exposure time. This can significantly improve the SceneCamera image in varying lighting conditions and where it is bright, such as outdoors. Note: Changing the exposure time with Auto or with an argument may affect the frame rate.
	<code>VPX_SendCommand("SceneA:videoShutter Auto");</code> <code>VPX_SendCommand("EyeA:videoShutter Report");</code> <code>VPX_SendCommand("EyeB:videoShutter 0.8");</code>



19.7.9 Dynamically Optimize Brightness and Contrast Settings

GUI: *Controls window > EyeA or EyeB tab > Video Group > AutoImage checkbox*

CLI : *videoAutoImage BoolValue*

EyeA:videoAutoImage BoolValue

EyeB:videoAutoImage BoolValue

Default: On

Continuously adjusts the video image brightness and contrast levels to optimal values for the specified eye.

Note 1: Only the region within the pupil scan area is examined, the pupil scan area rectangle must be of sufficient size for the algorithm to sample a range of gray levels, otherwise the algorithm will fail.

Note 2: The algorithm is under development and may change without notice.

```
VPX_SendCommand( "videoAutoImage On" ); // sets Eye A  
VPX_SendCommand( "EyeA:videoAutoImage On" );  
VPX_SendCommand( "EyeB:videoAutoImage On" );
```



Arrington Research

19.7.10 Adjust Pupil Scan Density

GUI: *Controls window > EyeA or EyeB tab > Threshold group > Pupil Scan Density slider*

CLI : `pupilScanDensity IntValue // sets Eye A`

`EyeA:pupilScanDensity IntValue`

`EyeB:pupilScanDensity IntValue`

IntValue: integer in range 1 to 20

Default: 7 with: `pupilSegmentationMethod Ellipse`

The value specifies the pixel sampling interval for the Threshold Segmentation Operation for the specified eye. The value 1 indicates to sample every pixel. The value 2 indicates to sample every other pixel in both x and y directions, so one fourth as many pixels are sampled with a setting of 2 as with a setting of 1. Etc.

Caution: Normally there is no need to sample very densely and doing so will greatly burden the CPU. For the GUI interface the slider has a default minimum value greater than 1 to avoid CPU overload. This default minimum may be changed with the CLI: `minimumPupilScanDensity`.

Note: previous versions provided for either a normalized floating point value in the range (0.0 – 1.0), or an integer in the range 1 – 20, however the normalized floating point values are no longer supported. There is no confusion with the value 1, because the minimum sampling interval of integer 1 and the maximum normalized density (1.0) are opposite ways of looking at the same thing.

See also: `minimumPupilScanDensity` `pupilSegmentationMethod`

```
VPX_SendCommand ( "pupilScanDensity 5" ); // says Eye A  
VPX_SendCommand ( "EyeA:pupilScanDensity 5" );  
VPX_SendCommand ( "EyeB:pupilScanDensity 5" );
```



Arrington Research

19.7.11 Override Pupil Scan Density Minimum

GUI:	<i>-none-</i>
CLI :	<code>minimumPupilScanDensity DensityIndex // sets Eye A</code> <code>EyeA:minimumPupilScanDensity DensityIndex</code> <code>EyeB:minimumPupilScanDensity DensityIndex</code> <i>DensityIndex: Integer in range 1 to 20, depending upon max density chosen.</i>
Default:	5 but depends upon <code>pupilSegmentationMethod</code>
	Overrides the minimum pupil scan density allowed on the Controls window > Threshold group > Pupil slider for the specified eye Fine sampling is rarely required and not generally recommended. <i>See also:</i> <code>pupilSegmentationMethod</code>
<p>WARNING: Setting the scan density too fine can create a huge burden on the CPU and possibly lock-out use of the GUI.</p>	
<pre>VPX_SendCommand ("minimumPupilScanDensity 12"); // sets Eye A VPX_SendCommand ("EyeA:minimumPupilScanDensity 12"); VPX_SendCommand ("EyeB:minimumPupilScanDensity 12");</pre>	



Arrington Research

19.7.12 Adjust Glint Scan Density

GUI: *Controls window > EyeA or EyeB tab > Threshold Group > Glint Scan Density slider*

CLI : `glintScanDensity IntValue // sets Eye A`

`EyeA:glintScanDensity IntValue`

`EyeB:glintScanDensity IntValue`

IntValue: integer in range 1 to 20

Default: **2 or 3**, depending on `glintSegmentationMethod`

The argument specifies the pixel sampling interval for the glint threshold segmentation operation for the specified eye. The value 1 indicates to sample every pixel. The value 2 indicates to sample every other pixel in both the x and y directions, so one fourth as many pixels are sampled as with a setting of 1. Etc.

The glint is usually much smaller than the pupil, so finer sampling is expected.

Caution: Normally there is no need to sample very densely and doing so will greatly burden the CPU. For the GUI interface the slider has a default minimum value greater than 1 to avoid CPU overload. This default minimum may be change with the CLI: `minimumGlintScanDensity`.

Note: Previous versions provided for either a normalized floating point value in the range (0.0 – 1.0), or an integer in the range 1 – 20; however the normalized floating point values are no longer supported. There is no confusion with the value 1, because the minimum sampling interval of integer 1 and the maximum normalized density (1.0) are opposite ways of looking at the same thing.

See also: `minimumGlintScanDensity` `glintSegmentationMethod`

`VPX_SendCommand ("glintScanDensity 5"); // sets Eye A`

`VPX_SendCommand ("EyeA:glintScanDensity 5");`

`VPX_SendCommand ("EyeB:glintScanDensity 5");`



19.7.13 Override Glint Scan Density Minimum

GUI:	<i>-none-</i>
CLI :	<code>minimumGlintScanDensity DensityIndex // Sets Eye A</code> <code>EyeA:minimumGlintScanDensity DensityIndex</code> <code>EyeB:minimumGlintScanDensity DensityIndex</code> <i>DensityIndex: Integer in range 1 to 20, depending upon max density chosen.</i>
Default:	1 or 3, depending on <code>glintSegmentationMethod</code>
	Overrides the minimum glint scan density allowed on the Controls window > Threshold group > Glint slider for the specified eye.
	Note: This is not normally required and small scan density values can over burden the CPU. See also: <code>glintScanDensity</code> <code>glintSegmentationMethod</code>
	<code>VPX_SendCommand ("minimumGlintScanDensity 3"); // sets Eye A</code> <code>VPX_SendCommand ("EyeA:minimumGlintScanDensity 3");</code> <code>VPX_SendCommand ("EyeB:minimumGlintScanDensity 3");</code>

19.8 EyeCamera Window

19.8.1 Adjust Pupil Scan Area	
GUI:	<i>EyeCamera window , EyeCamera toolbar , Top button</i> <i>Drag mouse in window to define the pupil scan area rectangle</i>
CLI :	pupilScanArea L T R B // sets eye A EyeA:pupilScanArea L T R B EyeB:pupilScanArea L T R B <i>L T R B: Normalized floating point values for the four sides.</i>
Default:	0.200 0.200 0.800 0.800
	Defines the scan area for the pupil of the specified eye. The four values are the floating point coordinates (0.0 – 1.0) of the bounding rectangle, listed in the order: Left, Top, Right, and Bottom.
	<pre>VPX_SendCommand ("pupilScanArea 0.3 0.2 1.0 0.4"); // sets eye A VPX_SendCommand ("EyeA:pupilScanArea 0.3 0.2 1.0 0.4"); VPX_SendCommand ("EyeB:pupilScanArea 0.3 0.2 1.0 0.4");</pre>

19.8.2 Pupil AutoCenter	
GUI:	<i>-none-</i>
CLI :	pupilAutoCenter BoolValue // sets eye A EyeA: pupilAutoCenter BoolValue EyeB: pupilAutoCenter BoolValue
Default:	Off
	Attempts to automatically keep the pupil in the center of the <i>EyeCamera</i> window. Only available on some camera models and in some videoModes. Note 1: The mappingFeature must be set to <i>Vector</i> . Note 2: Requires digital camera (e.g. USB-220). Note 3: The algorithm is under development and may change without notice. See also: mappingFeature
	<pre>VPX_SendCommand ("mappingFeature Vector; pupilAutoCenter ON");</pre>

19.8.3 Specify Pupil Scan Area Shape

GUI:	<i>-none-</i>
CLI :	<p>pupilScanShape <i>ScanShapeType</i> // sets Eye A</p> <p>EyeA:pupilScanShape <i>ScanShapeType</i></p> <p>EyeB:pupilScanShape <i>ScanShapeType</i></p> <p><i>ScanShapeType</i>: Rectangle, Ellipse.</p>
Default:	Elliptical
	<p>Specifies whether to change the scan area for the pupil to either rectangular or elliptical for the specified eye. Elliptical scan area is effective at eliminating dark spots that the software interprets as a pupil,</p> <p>Note: The overlay graphics do not change; the bounding rectangle for the ellipse or for the rectangle itself is drawn and appears the same. To see the ellipse (i) show the threshold dots, (ii) maximize/minimize the threshold, (iii) make entire image dark.</p>
<pre>VPX_SendCommand("pupilScanShape Ellipse"); // sets Eye A VPX_SendCommand("EyeA:pupilScanShape Ellipse"); VPX_SendCommand("EyeB:pupilScanShape Ellipse");</pre>	

19.8.4 Pupil and Glint Oval Fit Constraints

GUI:	<i>-none-</i>
CLI :	<p>pupilConstrained <i>BoolValue</i> // Sets Eye A</p> <p>EyeB:pupilConstrained <i>BoolValue</i></p> <p>glintConstrained <i>BoolValue</i></p> <p>EyeB:glintConstrained <i>BoolValue</i></p>
Default:	Yes (in most versions)
	<p>Controls whether or not the Pupil (glint) ellipse fit is allowed to extend beyond the scan area rectangle in the <i>EyeSpace</i> window.</p>
<pre>VPX_SendCommand("pupilConstrained False"); VPX_SendCommand("glintConstrained True");</pre>	



Arrington Research

19.8.5 Define Glint Scan Area

GUI:	<i>EyeCamera window , EyeCamera toolbar , Third button</i> <i>Drag mouse in window to define the glint scan area rectangle</i>
CLI :	<i>glintScanSize width height // Eye A</i> <i>EyeA:glintScanSize width height</i> <i>EyeB:glintScanSize width height</i> <i>width height : Normalized floating point coordinates.</i>

Default:	0.400 0.200
----------	-------------------------

Defines the width and height of the *Glint Scan Area* for the specified eye.
The two values are the normalized floating point width (X) and height (Y) values of the scan area bounding rectangle. The glint scan area rectangle is centered at the *glintScanOffset* vector.

```
VPX_SendCommand ( "glintScanSize 0.4 0.3" ); // sets Eye A  
VPX_SendCommand ( "EyeA:glintScanSize 0.4 0.3" );  
VPX_SendCommand ( "EyeB:glintScanSize 0.4 0.3" );
```

19.8.6 Define Offset of Glint Scan Area Relative to the Pupil

GUI:	<i>-none-</i>
CLI :	<i>glintScanOffset X Y // sets eye A</i> <i>EyeA:glintScanOffset X Y</i> <i>EyeB:glintScanOffset X Y</i> <i>X Y: Normalized floating point coordinates.</i>

Default:	0.010 0.080
----------	-------------------------

Defines offset of the *Glint Scan Area* relative to the center of the pupil.
The two values are the normalized coordinates of the offset vector from the center of the pupil to the center of the Glint Scan Area.

See also: *video_yokedGlint NO; glintScanUnYokedOffset*

```
VPX_SendCommand ( "glintScanOffset -0.05 0.1" ); // sets eye A  
VPX_SendCommand ( "EyeA:glintScanOffset -0.05 0.1" );  
VPX_SendCommand ( "EyeB:glintScanOffset -0.05 0.1" );
```



Arrington Research

19.8.7 Unyoke Glint Scan Area from the Pupil

GUI: *-none-*

CLI :
video_yokedGlint BoolValue //sets EyeA
EyeA:video_yokedGlint BoolValue
EyeB:video_yokedGlint BoolValue
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: **Yes**

Allows the Glint Scan Area to be unyoked from the pupil of the specified eye. i.e. the Glint Scan Area does not move with the pupil. Special applications only.

See also: [glintScanUnYokedOffset](#) [glintScanOffset](#)

```
VPX_SendCommand( "video_yokedGlint On" ); // Eye A  
VPX_SendCommand( "EyeA:video_yokedGlint On" );  
VPX_SendCommand( "EyeB:video_yokedGlint On" );
```

19.8.8 Define Offset of Unyoked Glint Scan Area

GUI: *-none-*

CLI:
glintScanUnYokedOffset X Y // Sets Eye A
EyeA:glintScanUnYokedOffset X Y
EyeA:glintScanUnYokedOffset X Y
X Y: Normalized floating point coordinates.

Defines offset of the *Glint Scan Area* relative to the upper left hand corner of the *EyeCamera* window for the specified eye.

The two values are the normalized coordinates of the offset vector.

See also: [video_yokedGlint](#)

```
VPX_SendCommand ( "glintScanUnYokedOffset 0.1 -3.0" );  
VPX_SendCommand ( "EyeA:glintScanUnYokedOffset 0.1 -3.0" );  
VPX_SendCommand ( "EyeB:glintScanUnYokedOffset 0.1 -3.0" );
```



Arrington Research

19.8.9 EyeCamera and EyeMovie Overlay Graphics Options

GUI:

-

CLI :

`eyeCameraOverlays +graphicsOption`

`eyeMovieOverlays +graphicsOption`

`graphicsOptions`: see here below.

Camera Default: `+CrossHair +ScanAreas +PupilFit +GlintFit +Warnings -Dots -Time -Frame -String -Width`

Movie Defaults: `-CrossHair -ScanAreas -PupilFit -GlintFit -Warnings -Dots +Time +Frame -String -Width`

<code>ALL</code>	: changes all overlays to show or not show
<code>CrossHair</code>	: the cross in the center of the pupil
<code>ScanAreas</code>	: rectangles where we look for the pupil or the glint
<code>PupilFit</code>	: the ellipse fit to the pupil
<code>GlintFit</code>	: the ellipse fit to the glint
<code>Warnings</code>	: the Quality warnings, e.g.: Glint Not Found, Pupil Oval Fit FAILED
<code>Dots</code>	: the scan threshold dots, same as <code>^D</code> , same as deprecated CLI: <code>showThresholdDots</code>
<code>Time</code>	: cumulative time stamp
<code>Frame</code>	: cumulative frame number of live video, OR the movie frame number of EyeMovie
<code>String</code>	: the string specified by CLI command: <code>EyeString "myString"</code>
<code>Width</code>	: shows Min & Max pupil width criteria, as when the sliders for these are moved

The above arguments may be immediately preceded with a plus sign (+) to enable them, a negative sign (-) to disable them, or a tilde (~) to toggle the current setting, e.g.:

`eyeCameraOverlays ~Dots`

`eyeMovieOverlays -ALL`

The `EyeA: EyeB: Both:` prefix may be used to specify the eye, e.g.: `EyeB:eyeCameraOverlays +dots`

Note that the `eyeCameraOverlays` command controls the overlays on both real-time camera video and on real-time analysis of EyeMovie video that is being played back. Any overlays that were painted into the video image when the EyeMovie was recorded (as specified with `eyeMovieOverlays`) are permanent and cannot be removed later during playback.

Note: `showThresholdDots Yes` is deprecated in favor of: `eyeCameraOverlays +dots`

```
VPX_SendCommand ( "eyeCameraOverlays +Frame +Time" ); // sets eye A
VPX_SendCommand ( "Both:eyeCameraOverlays +Dots" );
//
VPX_SendCommand ( "eyeCamera_Overlays +string; eyeString 'Astrid' " );
```



Arrington Research

19.8.10 EyeCamera Tool Bar Display

GUI: *-none-*

CLI : *videoToolBar Bool/Value // sets Eye A*

EyeA:videoToolBar Bool/Value

EyeB:videoToolBar Bool/Value

Bool/Value: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: **Yes**

Specifies whether to display the *EyeCamera* window ToolBar for the specified eye.

```
VPX_SendCommand( "videoToolBar off"); // sets eye A
```

```
VPX_SendCommand( "EyeA:videoToolBar off");
```

```
VPX_SendCommand( "EyeB:videoToolBar off");
```

19.9 Slip Compensation

These are relevant only when the `mappingFeature` method is `SlipComp`, see [19.7.1](#).

19.9.1 Slip Compensation Speed	
GUI:	<code>-none-</code>
CLI :	<code>slipComp_speed floatValue // sets Eye A</code> <code>EyeA: SlipComp_speed floatValue</code> <code>EyeB: SlipComp_speed floatValue</code> <code>floatValue: in range [0.005 1.0]</code>
Default:	0.03
<p>This is the speed at which the calculated error value due to slip is modified. A value of 1.0 causes immediate response to any error -- it will not filter vibrations and it will not filter transient glint loss, which are among the primary advantages of the <i>SlipCompensation</i> method. A value of 0.5 would use half of the current value and half of the previous average error based on the <i>Exponential Moving Average</i> (EMA) algorithm. A small parameter value is suitable for very slow slip. Because the EMA is recalculated with every new eye image, even a small value can produce an effect reasonably quickly. To get a feel for the amount of correction, try moving (slipping) the EyeCamera deliberately while fixating on the POG overlay spot and see how quickly it recovers.</p> <p>See also: <code>mappingFeature</code>, <code>slipComp_xGain</code>, <code>slipComp_yGain</code></p> <pre>VPX_SendCommand("slipCompSlipComp_speed 0.51"); // sets eye A VPX_SendCommand("EyeA:SlipComp_speed 0.1"); VPX_SendCommand("EyeB:SlipComp_speed 0.2");</pre>	

19.9.2 Slip Compensation X-Gain & Y-Gain	
GUI:	<code>-none-</code>
CLI :	<code>SlipComp_xGain floatValue // sets Eye A</code> <code>EyeA: SlipComp_xGain floatValue</code> <code>EyeB: SlipComp_yGain floatValue</code> <code>floatValue: in range: Real</code>
Default X:	1.0, i.e. no gain
Default Y:	1.1, i.e. slight gain
<p>Given a certain (slowly varying) calculated slip error, this controls how much that error value affects the gaze point in the (x) horizontal or (y) vertical. Default is 1.0, i.e. no gain as HMD slippage is typical in the vertical direction.</p> <p>See also: <code>mappingFeature</code>, <code>slipComp_speed</code>, <code>slipComp_yGain</code></p>	

19.10 EyeMovie related controls

19.10.1 Open / Close new EyeMovie file.	
GUI:	<i>File > EyeMovie > Unique EyeMovie Recording</i> <i>File > EyeMovie > New EyeMovie Recording ...</i> <i>File > EyeMovie > Close EyeMovie Recording</i>
CLI :	<code>eyeMovie_NewUnique</code> <code>eyeMovie_NewName fileNameString</code> <code>eyeMovie_Close</code>
Default:	<i>NewUnique creates a file name from the data & time</i>
Open/Close a new EyeMovie and starts recording. Version 2.9.3.123 moves this menu to main <i>File</i> menu, was in each <i>EyeCamera</i> window popup menus.	
<code>VPX_SendCommand("eyeMovie_NewName 'my eye movie' ");</code>	

19.10.2 EyeMovie Play	
GUI:	<i>File > EyeMovie > Play (toggle)</i>
CLI :	<code>eyeMovie_Play BoolValue</code> <i>BoolValue:</i> Yes, No, True, False, On, Off, 1, 0, Toggle
Default:	<i>Off</i>
Toggles between live EyeCamera video and EyeMovie video. If no EyeMovie has been loaded, they the Open EyeMovie Dialog Box is shown. See also: eyeMovie_Load	
<code>VPX_SendCommand("eyeMovie_Used Toggle");</code>	



Arrington Research

19.10.3 EyeMovie Load ...

GUI: *File > EyeMovie > Load EyeMovie ...*

CLI :
`eyeMovie_Load fileNameString`
`eyeMovie_Load // with no argument will show the Load EyeMovie DialogBox`
`eyeMovie_LoadDialog`
`eyeMovie_UnLoad`

Default:

Loads the specified EyeMovie (must specify extension), or if no argument is given the Open EyeMovie DialogBox is shown.

NOTE: this does NOT start the movie playing, use `eyeMovie_Play On` to start playing.

Version 2.9.3.1213 launches Load Dialog if no parameter, previously gave an error.

`VPX_SendCommand("eyeMovie_Load 'myEyes.AVI' ");`

19.10.4 EyeMovie EndAction

GUI: *File > EyeMovie > EndAction : { Stop, Loop, Reverse }*

CLI :
`eyeMovie_EndAction Option`
Option: any one of: Stop, Loop, Reverse

Default: *Reverse – this reduces discontinuities*

Specifies what to do when the movie ends.

`VPX_SendCommand("eyeMovie_EndAction Reverse");`

19.10.5 EyeMovie Percent Location

GUI: *Controls window, EyeA tab, Frame 0% (slider) #0*

CLI :
`eyeMovie_Percent normalizedValue`
normalizedValue: in { 0.0 ... 1.0 }

Default: *Starts at the beginning = 0.0*

Allows user to skip to specified location in the movie.

`VPX_SendCommand("eyeMovie_Percent 0.5"); // go to middle`



Arrington Research

19.10.6 EyeMovie Play Speed

GUI: *Controls window, EyeA tab, Speed 1.0 (slider)*

CLI : *eyeMovie_Speed non-zero realNumber*
non-zero realNumber: in { 0.0 ... 99.0 }

Default: *1.0*

Allows user to control the playback speed of the movie. The default 1.0 attempts to be normal, or a reasonable speed. Less than 1.0 is slower, more than 1.0 is faster.

```
VPX_SendCommand( "eyeMovie_Speed 0.5"); // go to middle
```

19.10.7 EyeMovie Binocular

GUI: *File > EyeMovie > EyeMovie is Binocular (check mark)*

CLI : *eyeMovie_IsBinocular BoolValue*
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: *No*

Tells *ViewPoint* to split the movie in half and sent the EyeB half to the EyeB EyeCamera window.

```
VPX_SendCommand( "eyeMovie_IsBinocular TRUE"); //
```

19.10.8 EyeMovie Binocular

GUI: *File > EyeMovie > EyeMovie is Binocular (check mark)*

CLI : *eyeMovie_IsBinocular BoolValue*
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: *No*

Tells *ViewPoint* to split the movie in half and sent the EyeB half to the EyeB EyeCamera window.

```
VPX_SendCommand( "eyeMovie_IsBinocular TRUE"); //
```



Arrington Research

19.10.9 EyeMovie Zoom / Pan

GUI: *EyeCamera window – Pan with mouse when Right-mouse button is down (some versions)*

CLI :
eyeMovie_Zoom { 0.0 to 0.99 } // zero is no zoom, 0.99 is only a few pixels
eyeMovie_Center {0.0 to 0.99} {0.0 to 0.99} // x and y movie location to center upon

Default: 0.0 and 0.0 0.0

Allows panning and zooming of the movie. You can only pan if you are zoomed in.

```
VPX_SendCommand( "eyeMovie_Zoom 0.5; eyeMovie_Center 0.2 0.8");
```

19.11 Video related controls

Video commands relating to the eyes take a *VideoStream Specifier Prefix*: **EyeA:** **EyeB:**, there should be no white space between the specifier and the command.

19.11.1 Specify EyeCamera Video Input Standard	
GUI:	<i>EyeCamera window > Monitor icon > Video Standard > { NTSC, PAL, SECAM }</i>
CLI :	videoStandard Option // Eye A EyeA:videoStandard Option EyeB:videoStandard Option <i>Option: any one of: NTSC, PAL, SECAM</i>
Default:	NTSC upon first run, but saved in preferences file thereafter.
	<p>Specifies which video mode to use for analog cameras. Note: This command does not apply to Digital cameras, such as in the USB-220 systems.</p> <p>The <i>Arrington Research</i> 60Hz analog systems always only include NTSC (color) or EIA (B&W) cameras. The <i>ViewPoint EyeTracker</i> allows for the use of other format (CCIR, PAL, and SECAM) cameras that may be provided in fMRI and other equipment from other manufacturers.</p> <p>Cameras used in the <i>ViewPoint EyeTracker</i> are in a closed system and are not affected by the television standard of the country in which they are used.</p> <p>Note: SilverBox (USB-60x3) currently only supports NTSC (and EIA).</p> <p><i>ViewPoint</i> tries to ensure that it starts up in a friendly way each time; to facilitate this the <i>videoStandard</i> selection is saved in a special Preferences file that is evaluated each time that <i>ViewPoint</i> is launched.</p> <p>EIA (Electronics Industry Association) is the black & white (B&W) television format in USA, Japan, and Canada. EIA is the predecessor to NTSC color standard. These are approximately 60 Hz. Specify NTSC as the <i>videoStandard</i> Option for NTSC and EIA cameras.</p> <p>CCIR (<i>Committee Consultatif International Radiotélécommuniqué</i>) is the B&W television standard used in Europe and Australia. CCIR is the predecessor to the PAS color standard. These are approximately 50 Hz. Specify PAS as the <i>videoStandard</i> Option for PAL and CCIR cameras.</p>
	<pre>VPX_SendCommand("videoStandard NTSC"); // also for EIA VPX_SendCommand("videoStandard PAL"); // also for CCIR</pre>



Arrington Research

19.11.2 Specify SceneCamera Video Input Standard

GUI: *-none-*

CLI : **scene_videoStandard Option**
Option: any one of: NTSC, PAL, SECAM

Default: **NTSC**

Specifies which video mode to use for the scene camera.

Note: SilverBox (USB-60x3) currently only supports NTSC.

Note: Does not apply to Digital cameras, such as in the USB-220 systems.

Implemented 2.8.5.017

```
VPX_SendCommand( "scene_VideoStandard NTSC" );
```

19.11.3 Specify Video Operation Mode

GUI: *EyeCamera window > Monitor Icon > Mode > Setup, Precision, Speed*

EyeCamera window > Monitor Icon > Mode > 90, 220 (USB-220 products)

CLI : **videoMode ProcessingMode** // sets Eye A

EyeA:videoMode ProcessingMode

EyeB:videoMode ProcessingMode

Analog ProcessingMode: Setup, Precision, Speed

Digital ProcessingMode: 90, 220, 350, 400 (USB-220 products)

Default: **Setup, 220**

Specifies which operation mode to use for the specified eye. The options vary depending upon the product that is used.

Analog video products (PC60, USB-60x3) allow three modes: Setup (320x240@30Hz), Precision (640x480@60Hz), and Speed (320x240@60Hz).

Note: in older versions Speed mode ran at 640x240@60 and Speed2 mode ran at (320x240@60Hz), but the former is obsolete.

Digital video products (USB-400, USB-220, and USB-90) allow modes: 90, 220, 350, and 400 that specify the frame rate, while the image size is maximized for the allowable bandwidth. The maximum allowed will depend upon the product license purchased.

Use *EyePrefix* to specify a particular eye. See [14.1](#) for more details.

```
VPX_SendCommand( "videoMode Precision" ); // sets Eye A
```

```
VPX_SendCommand( "EyeA:videoMode Precision" );
```

```
VPX_SendCommand( "EyeB:videoMode 90" );
```



Arrington Research

19.11.4 Specify Dark or Bright Pupil Tracking

GUI: *EyeCamera window > Monitor Icon > Pupil Type > Dark Pupil, Bright Pupil*

CLI :
pupilType *PupilType* // Eye A
EyeA:pupilType *PupilType*
EyeB:pupilType *PupilType*
PupilType: Dark, Bright

Specifies dark or bright pupil tracking for the specified eye. When dark pupil is selected, the tracking algorithm looks for pixels below threshold. When bright pupil is selected, the tracking algorithm looks for pixels above threshold.

```
VPX_SendCommand( "pupilType dark"); // Eye A
VPX_SendCommand( "EyeA:pupilType dark");
VPX_SendCommand( "EyeB:pupilType dark");
```

19.11.5 Specify Pupil Segmentation Method

GUI: *EyeCamera window > Monitor Icon > Pupil Segmentation Method >*
{ Centroid, Oval Fit, Ellipse }

CLI :
pupilSegmentationMethod *Method* // Eye A
EyeA:pupilSegmentationMethod *Method*
EyeB:pupilSegmentationMethod *Method*
Method: Centroid, OvalFit, Ellipse

Default: **OvalFit (for older versions), Ellipse (for newer versions)**

Specifies which pupil segmentation method to use for the specified eye. The Centroid is obtained first in all cases. OvalFit or Ellipse perform additional image processing and fitting.

The pupil width and pupil aspect ratio will not be available unless either *OvalFit* or *Ellipse* is selected.

For more information on the PupilSize relative to which pupil segmentation method is selected, see [VPX_GetPupilSize](#).

Note: Blinks are detected by examining when the pupil aspect ratio is below criterion; this calculation usually performs better with the older OvalFit than the newer Ellipse method.

```
VPX_SendCommand( "pupilSegmentationMethod Ellipse"); // Eye A
VPX_SendCommand( "EyeA:pupilSegmentationMethod Ellipse");
VPX_SendCommand( "EyeB:pupilSegmentationMethod Ellipse");
```



Arrington Research

19.11.6 Specify Glint Segmentation Method

GUI: *EyeCamera window > Monitor Icon > Glint Segmentation Method >*
 { Centroid, Oval Fit }

CLI : *glintSegmentationMethod Method // Eye A*
 EyeA:glintSegmentationMethod Method
 EyeB:glintSegmentationMethod Method
 Method: Centroid, OvalFit

Default: **OvalFit**

Specifies which *Glint Segmentation Method* to use for the specified eye, either Oval Fit or Centroid. Actually the Centroid is obtained in either case, but Oval Fit performs additional image processing and fitting.

Note: changing this may also change the values of *minimumGlintScanDensity* and *glintScanDensity*.

```
VPX_SendCommand( "glintSegmentationMethod OvalFit" ); // Eye A  
VPX_SendCommand( "EyeB:glintSegmentationMethod OvalFit" );
```

19.11.7 Toggle Freeze Video Image Preview On / Off

GUI: *EyeCamera window, SnowFlake button* **^F**

CLI : *videoFreeze BoolValue // EyeA*
 videoFreezeSync BoolValue // Synchronizes EyeA & EyeB toggle effect
 EyeA:videoFreeze BoolValue
 EyeB:videoFreeze BoolValue
 BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: **No**

Allows simultaneous freezing and unfreezing of all video images (EyeA, EyeB, Scene, etc.), *PenPlot* window display and eye data collection.

See also: *dataFile_Pause Toggle*

Use *videoFreeze* to freeze a single target video channel.

```
VPX_SendCommand( "videofreeze Yes" ); // Eye A  
VPX_SendCommand( "EyeB:videofreeze Yes" );  
FKey_cmd 1 {videoFreezeSync Toggle}
```



Arrington Research

19.11.8 Reset Video Capture Device

GUI: *EyeCamera window > monitor icon > Reset EyeCamera Video*
 Stimuli > ViewSource > Reset SceneCamera
CLI : videoReset // Eye A
 sceneReset

Resets the specified video capture device for the specified eye or scene video.

Prefix VideoStream specifier: EyeA: EyeB:

```
VPX_SendCommand( "videoReset" );  
VPX_SendCommand( "EyeB:videoReset" );  
VPX_SendCommand( "sceneReset" );
```

19.11.9 VideoMirror

GUI: -*none*-
CLI : videoMirror axisFlags
 axisFlags: H , V

Mirrors (flips) the video image (e.g: [b]) Horizontally (left-right: [d]) and or Vertically (up-down: [P]).

Both will flip the [b] to a [q]. Arguments must be separated by white spaces.

Currently works only with certain hardware, e.g.: USB-220.

Prefix VideoStream specifier: EyeA: EyeB:

Version 2.9.3.121 changed CLI name from 'mirror' to 'videoMirror'.,

See also: calibration_InitWithHorizontalFlip, calibration_InitWithVerticalFlip

```
VPX_SendCommand( "videoMirror V H" );      // VH is invalid, must separate arguments.  
VPX_SendCommand( "EyeB:videoMirror H" ); // Omission deletes, so -V is implied.
```

19.12 Calibration Controls

It should be noted that some of these commands are closely inter-related and changing one may have the side effect of changing others. Calibration is described in [Chapter 8](#).

19.12.1 Start Auto-Calibration	
GUI:	<i>EyeSpace window > Auto-Calibrate button</i> ^A (toggle)
	<i>EyeSpace window > STOP Calibration button</i> ^A (toggle)
CLI :	calibrationStart
	calibrationStop
Starts / Stops the automatic calibration process. While autocalibration is in progress, the <i>Auto-Calibrate button</i> name changes to <i>STOP Calibration</i> . Pressing <i>STOP Calibration</i> immediately terminates the automatic calibration process, before its normal completion.	
<code>VPX_SendCommand ("calibrationStart");</code> <code>VPX_SendCommand ("calibrationStop");</code>	

19.12.2 Specify Calibration Stimulus Presentation Speed	
GUI:	<i>EyeSpace window > Advanced button > Duration slider</i>
CLI :	calibration_StimulusDuration milliseconds milliseconds : integer between 1 and 400
Default:	depends upon the OS
Specifies the delay in milliseconds between calibration stimulus changes (zoom rectangle decrements). <i>ViewPoint</i> attempts to set an optimal default value according to the operating system version. The “milliseconds” specification is only approximate and unfortunately varies between Microsoft operating system versions. This value also affects the warning time and the Inter-Stimulus Interval (ISI) that specify durations in units of stimulus duration milliseconds. The maximum duration is 400 ms, and the minimum is 1 millisecond.	
<code>VPX_SendCommand ("calibration_StimulusDuration 145");</code>	



Arrington Research

19.12.3 Specify the Duration of Calibration Warning Notice

GUI:	<i>EyeSpace window, Advanced button, Warning slider</i>
CLI :	calibration_WarningTime durationUnits <i>durationUnits</i> : integer between 0 and 100
Default:	20
Specifies the delay for posting a warning that calibration is about to start. <i>ViewPoint</i> attempts to set an optimal default value according to the operating system version. The time is only approximate.	
The delay is in units of the stimulus duration specified by the command: calibration_StimulusDuration . If the calibration Stimulus Duration is set to 50 and the Warning Time is set to 20, then there will be $20 * 50$ millisecond delays that would total roughly 1 second.	
The value 0 specifies no warning is to be given.	
<i>See also:</i> calibration_StimulusDuration	
VPX_SendCommand ("Calibration_WarningTime 15");	

19.12.4 Specifies Interval Between Presentation of Calibration Points

GUI:	<i>-none-</i>
CLI :	calibration_ISI durationUnits <i>durationUnits</i> : integer between 1 and 9
Default:	2
Specifies the inter-stimulus interval between calibration points. <i>ViewPoint</i> attempts to set an optimal default value according to the operating system version. The time is only approximate.	
The delay is in units of the stimulus duration specified by the command: calibration_StimulusDuration . If the calibration Stimulus Duration is set to 50 and the ISI is set to 20, then there will be $20 * 50$ millisecond delays that would total roughly 1 second.	
The value 0 specifies no ISI time.	
<i>See also:</i> calibration_StimulusDuration	
VPX_SendCommand ("Calibration_ISI 2");	



Arrington Research

19.12.5 Specify Number of Calibration StimulusPoints

GUI: *EyeSpace window > drop-down list: 6 ... 72*

CLI : *calibration_Points numberOfPoints*
numberOfPoints : integer from the following set:
{ 6, 9, 12, 16, 20, 25, 30, 36, 42, 49, 56, 64, 72 }

Default: **16**

Sets the number of calibration points to be presented.

IMPORTANT: 6 calibration points should not be used unless the subject cannot maintain focus long enough to calibrate additional points. 6 points only has 2 vertically spaced points, and thus gives a poor vertical calibration. 9 points is the minimum recommended points.

Note that the valid values are either N x N or N x (N-1)

Six points should generally never be used, because this provides only horizontal calibration.

See manual calibration option if fewer points are required.

`VPX_SendCommand ("calibration_Points 12");`

19.12.6 Specify Calibration StimulusPoint Color

GUI: *EyeSpace window > Advanced button > Set Stimulus Color button*

CLI : *calibration_StimulusColor intRed intGreen intBlue*
intRed 0-255 *intGreen* 0-255 *intBlue* 0-255

Specifies the red, green and blue components (0 to 255) of the calibration StimulusPoints E.g. 255 255 255 is white and 100 100 255 is a sky blue.

`VPX_SendCommand ("calibration_StimulusColor 255 255 025");`

19.12.7 Specify Calibration Stimulus Window Background Color

GUI: *EyeSpace window > Advanced button > Set Background Color button*

CLI : *calibration_BackgroundColor intRed intGreen intBlue*
intRed 0-255 *intGreen* 0-255 *intBlue* 0-255

Specifies the red, green and blue components (0 to 255) of the calibration *Stimulus* window background E.g. , 255 255 255 is white and 100 100 255 is a sky blue.

`VPX_SendCommand ("calibration_BackgroundColor 050 100 255");`



Arrington Research

19.12.8 Specify CalibrationStimulus Type

GUI: *-none-*

CLI : *calibration_StimulusType <options>*
<options> include: Shrink, Bounce

Default: **Bounce**

Shrink is the original type where a rectangle shrinks to a dot and disappears.

Bounce is similar, but it re-expands to original size

`VPX_SendCommand ("calibration_StimulusType Bounce");`

19.12.9 Calibration Snap Mode

GUI: *EyeSpace window > Advanced button, Snap Presentation Mode checkbox*

CLI : *calibration_SnapMode BoolValue*
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Set **ON** if *viewSource SceneCamera*, otherwise set **OFF**

SnapMode means that the Calibration Stimulus Images (the zooming concentric rectangles) are not presented, and the calibration of the currently selected point is immediately performed based on the current eye position. This is useful for remote or manual calibration as when using the *SceneCamera* option and when performing calibration with StimulusPoints that are controlled and generated on a remote computer.

This command affects the behavior of the *Re-Present button* [19.12.26](#) and the *Slip-Correction button* [19.12.23](#) buttons/commands and is indicated by the appearance of an asterisk (*) on these buttons when in this special mode.

SnapMode only effects the GUI interface, it does not effect CLI commands such as *calibration_Snap*.

See also:

calibration_Snap

calibration_AutoIncrement

calibrationRedoPoint

`VPX_SendCommand ("calibration_SnapMode ON");`



Arrington Research

19.12.10 Calibration Beep when Finished

GUI: *EyeSpace window > Advanced button, Beep Finished checkbox*

CLI : *calibration_BeepFinished BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Set **ON** if **viewSource SceneCamera**, otherwise set **OFF**

This is very useful when the user is calibrating the subject with the *SceneCamera* system, where the user is positioning a laser pointer, or a finger and pressing a button. The beep notifies the user that the entire calibration sequence is complete. Otherwise, continued snap&inc (default FKey 8) will continue and recycle through the calibration points.

```
VPX_SendCommand ( "calibration_StimulusType Bounce" );
```

19.12.11 RePresent in Snap Calibration Mode

GUI: *EyeSpace window > Re-present * (when asterisk is showing)*

CLI : *calibration_Snap*

calibration_Snap integer

snap&inc

Default:

The calibration of the currently selected point is immediately performed based on the current eye position without presenting the Calibration Stimulus Images (the zooming concentric rectangles).

Compare this to: *calibrationRedoPoint* that does show the stimuli.

This is useful for remote or manual calibration as when using the *SceneCamera* option and when performing calibration with StimulusPoints that are controlled and generated on a remote computer.

The command *snap&inc* provides a handy way to { *calibration_Snap*; *calibration_SelectPoint NEXT* }

Use of this command does not require, and in many situations obviates the need for, changing the *calibration_SnapMode*.

See also: *calibration_SelectPoint NEXT*, *calibration_SelectIndex*, *calibrationRedoPoint*

KnownIssue: once *calibration_Snap N* is specified, subsequent *calibration_Snap* (without an argument) use the last specified N rather than the currently selected point.

```
VPX_SendCommand ( "calibration_Snap" );
```

```
VPX_SendCommand ( "calibration_Snap %d", pn );
```



Arrington Research

19.12.12 AutoIncrement Calibration Mode

GUI: *EyeSpace window > Advanced button > Auto-Increment checkbox*

CLI : *calibration_Autoincrement BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Set **ON** if *viewSource SceneCamera*, otherwise set **OFF**

Auto-Increment affects the behavior of the *EyeSpace window > Re-Present button* and CLI: *calibration_Snap* command. It is indicated by a double plus (**++**) on the button when in this special mode.

This is useful to turn this OFF when trying to repeatedly capture an untrained animal's gaze at one particular point. This mode is useful for manual calibration as with a *SceneCamera*.

`VPX_SendCommand ("calibration_AutoIncrement TRUE ");`

Presentation Order

19.12.13 Calibration StimulusPoint Presentation Order

GUI: *EyeSpace window > Advanced button > Presentation Order pulldown list*

CLI : `calibration_PresentationOrder orderChoice`

`orderChoice: Sequential, Random, Custom`

This affects both auto-calibration and manual calibration that sequence through the calibration index values. If the user has set: `calibration_PresentationOrder Sequential` then the index value and the calibration StimulusPoint number are the same.

This selection automatically changes as a side effect when changing the calibration *PointLocationMethod* selection (see section [19.12.19](#)), as follows:

<i>Point Locations</i>	<i>Presentation Order</i>
Automatic	Random
Custom Set	Random
OnContent	Sequential

`VPX_SendCommand ("calibration_PresentationOrder Random");`

19.12.14 Specify Calibration StimulusPoint Presentation Order

GUI: `-none-`

CLI : `calibration_CustomOrderList n1 n2 n3 n4 n5 n6 ...`

Default: `6 5 4 3 2 1 9 8 7 12 11 10 16 15 14 13 20 19 18 17 21 22 23 ...`

Allow specification of up to 72 calibration point in the desired order of presentation.

Caution: An error will occur if the user has provided a point number in the list, which is larger than the selected number of calibration Points (i.e., the quantity of calibration points). For example, if the user specified: `customOrderList 6 72 4 69 2 1` and also specified: `calibrationPoints 12`, the points 72 and 69 would cause errors, because they are greater than 12.

See also:

`calibration_SelectIndex, calibration_CustomPoint`

`calibration_SelectPoint, calibrationPoints`

`VPX_SendCommand ("calibration_CustomOrderList 6 1 5 2 4 3");`



Arrington Research

19.12.15 Specify Individual Custom Calibration StimulusPoints

GUI: *-none-*

CLI : *calibration_CustomOrderEntry index calStimPoint*
index : integer in { 1 to N }
calStimPoint : integer in { 1 to N }

Default: See [calibration_CustomOrderList](#), just above.

Used to specify individual custom calibration point entries.

Caution: see the Caution section in [calibration_CustomOrderList](#), above.

See also:

[calibration_SelectIndex](#)

[calibration_SelectPoint](#)

```
VPX_SendCommand ( "calibration_CustomOrderEntry 1 6" );
```

19.12.16 Display Custom Calibration StimulusPoint Order

GUI: *-none-*

CLI : [calibration_CustomOrderDump](#)

Prints the customOrder of calibration points to the *History* window.

```
VPX_SendCommand ( "calibration_CustomOrderDump" );
```

19.12.17 Select the Specified Calibration DataPoint

GUI: *EyeSpace window, DataPoint slider*

or/ Click mouse in EyeSpace window, calibration graphics well

CLI : [calibration_SelectPoint pointSelection](#)

pointSelection: LAST, NEXT, or int { 1 .. N }

Selects the specified calibration *DataPoint*. The point will be highlighted in the *EyeSpace* window. This is the same number as the value shown next to the [EyeSpace window > Data Point slider](#).

Note: [Controls window > Display tab > Calib Region checkbox](#) will show the calibration *StimulusPoint* locations.

See also: [calibration_SelectIndex](#)

```
VPX_SendCommand ( "calibrationSelectPoint 7" );
```



Arrington Research

19.12.18 Specify an Index Number for Calibration DataPoint

GUI: *-none-*

CLI : **calibration_SelectIndex indexSelection**
 indexSelection: LAST, NEXT, or int { 1 .. N }

Selects the *SequenceIndex* number that maps to the specified calibration *DataPoint* (as shown on the slider). The corresponding *DataPoint* will be highlighted in the *EyeSpace* window and to the [EyeSpace window > Data Point slider](#) will be adjusted to show the *DataPoint*.

When we set **calibration_PresentationOrder Sequential** the selected index and the selected point are the same. When **calibration_PresentationOrder** is Random or Custom, the index is incremented and the *DataPoint* is taken from the random or custom table.

Note: if the user has set: **calibration_PresentationOrder Random** then the series is re-randomized every time the set finishes, so that there is a new set for the next loop.

Note: this index is not displayed anywhere.

See also:

[calibration_SelectPoint](#)

[calibration_PresentationOrder](#)

[calibration_CustomOrderList](#)

[calibration_CustomOrderEntry index calStimPoint](#)

```
VPX_SendCommand ( "calibration_SelectIndex 7" );
// look up Pt# in CustomOrderList
```

Point Locations

19.12.19 Calibration StimulusPoint Location Method			
GUI:	<i>EyeSpace window > Advanced button > Point Locations pulldown list</i>		
CLI :	calibration_PointLocationMethod <i>methodOption</i> <i>methodOption</i> includes { Automatic, OnContent, Custom }		
Default:	Automatic.		
<i>methodOption</i>	Custom	OnContent	AutomaticGrid
Presentation Order	Random	Sequential	Random
Point Locations	Custom Set	OnContent	Automatic
<i>Button Name</i>	Adjust Custom Points	Adjust Calibration Points	Adjust Calibration Area
Auto-Increment	Off	Checked ON	Off
Snap Presentation	Off	Checked ON	Off
Beep Finished	-unchanged	Checked ON	-unchanged
Show GridLines	-unchanged	-unchanged	-unchanged
Controls Window	-unchanged	Regions tab selected	Regions tab selected
Controls: Regions tab	Nothing	Calibrate On Content	Nothing
Note: This replaces both calibration_OnContent <BoolValue> and calibration_CustomPointsUsed <BoolValue>			
These two legacy functions currently do as expected with an ON argument, and set calibration_PointLocationMethod AutomaticGrid with an OFF argument.			
<i>See also:</i>			
calibration_SelectIndex, calibration_SelectPoint, calibration_CustomOrderList, calibration_CustomPoint calibration_CustomOrderEntry, calibration_CustomOrderDump			
VPX_SendCommand ("calibration_CustomOrderEntry 1 6");			



19.12.20 Specify Custom Calibration StimulusPoint Locations

GUI: *-none-*

CLI : `calibration_CustomPoint indexNumber xLoc yLoc`
 indexNumber is an integer from 1 to the number of points used.
 xLoc, yLoc are floating point numbers for the normalized location of the point.

Specifies custom locations of the calibration *StimulusPoints*. This is useful when avoiding occluding objects and for *Partial Binocular Overlap* (PBO) situation. These points will only be used if you set: `calibration_PointLocationMethod Custom`.

Note: The nearest-neighbor grid-lines in the *EyeSpace* are not automatically draw when this option is used, because the points could be in any configuration.

See also:

`calibration_PointLocationMethod Custom` ; `calibration_PointDump`

`calibration_CustomOrderList` ; `calibration_ShowEyeSpaceGrid`; `stereoSwapEyes`

```
stereoDisplay ON
//
calibration_Points 9
calibration_PointLocationMethod Custom
calibration_PresentationOrder Sequential
//
// EYE_A
EyeA:calibration_CustomPoint 1 0.1 0.05
EyeA:calibration_CustomPoint 2 0.1 0.5
EyeA:calibration_CustomPoint 3 0.1 0.95
//
.....
//
// EYE_B
EyeB:calibration_CustomPoint 1 0.1 0.15
EyeB:calibration_CustomPoint 2 0.1 0.5
EyeB:calibration_CustomPoint 3 0.1 0.85
...
```



Arrington Research

19.12.21 Dump Custom Calibration StimulusPoints Locations

GUI: [-none-](#)

CLI : [calibration_PointDump](#)

Prints the calibration point list in the *History* window.

See also:

[calibration_CustomPoint](#)

[calibration_PointLocationMethod Custom](#)

[Calibration_PointDump](#)

19.12.22 Display Nearest-Neighbor Gridlines in *EyeSpace* Window

GUI: [Advanced Calibration window, Show GridLines on EyeSpace checkbox](#)

CLI : [calibration_showEyeSpaceGrid BoolValue](#)

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: [Yes](#)

Turns ON or OFF the “spider web” nearest-neighbor grid between calibration points in the *EyeSpace* window.

See also: [calibration_PointsUsed](#)

[calibration_showEyeSpaceGrid NO](#)

19.12.23 Compensate for Slip

GUI: [EyeSpace window > Slip-Correction button](#)

CLI : [calibrationSlipCorrection CalibrationPoint](#)

CalibrationPoint: Index number of the point to slip correct.

Re-presents the specified calibration DataPoint and re-aligns calibration to compensate for slip.

Normally a point near the center of the display should be chosen.

See also:

[calibration_SnapMode](#)

`VPX_SendCommand ("calibrationSlipCorrection 7");`



Arrington Research

19.12.24 Adjust Calibration Area

GUI:	<i>Controls window > Regions tab > Calibration Region radio button Then use the mouse in the GazeSpace window to drag out the rectangular region</i>
CLI :	Calibration_RealRect L T R B <i>L T R B: Normalized floating point values for the four sides.</i>
Default:	0.1 0.1 0.9 0.9
Allows the user to adjust the calibration area (size and position) within which the calibration <i>StimulusPoints</i> are presented. The four values are the floating point coordinates (0.0 – 1.0) of the bounding rectangle, listed in the order: Left, Top, Right and Bottom. For GUI control, the user must use the mouse in the <i>GazeSpace</i> window to drag out the rectangular region.	
<code>VPX_SendCommand ("calibration_RealRect 0.2 0.2 0.8 0.8");</code>	

19.12.25 Undo the Last Operation on a Calibration DataPoint

GUI:	<i>EyeSpace window > Undo button</i>
CLI :	calibrationUndo
Re-centers the selected calibration point.	
<code>VPX_SendCommand ("calibrationUndo");</code>	

19.12.26 Re-Present the Specified Calibration DataPoint

GUI:	<i>EyeSpace window > Re-present button</i>
CLI :	calibrationRedoPoint CalibrationPoint <i>CalibrationPoint: integer number of the point to re-present.</i>
Presents or Re-presents the current or specified calibration DataPoint <u>using</u> the Calibration Stimulus Images (the zooming concentric rectangles). The <i>CalibrationPoint</i> argument is optional, if omitted, the currently selected calibration point will be presented. The behavior is <u>NOT</u> modified by the calibration_SnapMode and calibration_AutoIncrement specifications. To immediately get the calibration point without any calibration graphics, use calibration_Snap . Note: The <i>CalibrationPoint</i> argument corresponds to the slider values in the <i>EyeSpace</i> window (not the <i>SequenceIndex</i>) and does not depend on the calibration_presentationOrder chosen. See also: calibration_SelectPoint NEXT , calibration_SelectIndex , calibration_Snap	
<code>VPX_SendCommand ("calibrationRedoPoint 7");</code>	
<code>VPX_SendCommand ("calibrationRedoPoint"); // defaults to current point</code>	



Arrington Research

19.12.27 Save Image of Eye at Each Calibration DataPoint

GUI: **-none-**

CLI : **calibration_SaveEyeImages BoolValue**

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: **On**

OBSOLETE: This was changed in the later 2.9 versions such that a matrix of calibration eye images is generated every time a calibration is performed. Separate image matrix files are saved for each eye. They can be viewed via the button in the *EyeSpace* window.

(Old behavior: Off by default; when enabled this command will take a snapshot of the eye during calibration stimulus presentation at each calibration point.)

This feature is useful to examine the position of the eye when presented the stimulus. This allows users to determine any problems with the subject's eyes or loss of a direct image of the pupil at a certain calibration point.

`VPX_SendCommand ("calibration_SaveEyeImages yes");`

19.12.28 Calibration Quality

GUI: **-none-**

CLI : **calibration_dumpQuality**

Default: **Off**

Dumps calibration quality information to the *History* window, including:

(a) Error for each point, (b) ssqErr for each method and each eye.

Here the error is the distance between the calibration *StimulusPoint* and the calculated Position of Gaze (POG), for each point.

Optional *EyePrefix* defaults to **Both**: if binocular and no prefix specified.

`VPX_SendCommand ("EyeA:calibration_dumpQuality");`



Arrington Research

19.12.29 Nudge

GUI:	<i>Mouse click in the GazeSpace window at the location at which the subject is known to be looking.</i>
CLI :	<code>gazeNudge xPos yPos gazeNudgeInc xlnc ylnc</code> <i>xPos, yPos, xlnc, ylnc: are floating point numbers</i>
Default	0.0
If the calibration appears to have slipped you may want to adjust it on the fly, without using the Slip-Correction procedure. Note that the gazeNudge is applied only to the <i>Corrected</i> data. xPos and yPos are the normalized position coordinates of the correct POG. xlnc and ylnc are the normalized position increments to be added to the nudge vector.	
<code>VPX_SendCommand ("gazeNudgeInc %g %g", x, y); VPX_SendCommand ("FKey_cmd 9 { gazeNudgeInc 0.2 0.0 }");</code>	

19.12.30 Manual Calibration

GUI:	<i>EyeSpace window > Advanced button Advanced window > Manual Calibration section</i>
CLI :	<code>manualCalibration_AspectRatio aspect manualCalibration_Center manualCalibration_Gain gain</code> <i>aspect, gain: floating point values in range 0.01 to 1.0</i>
Default	0.0
Non-verbal subjects may be calibrated manually by specifying (a) the aspect ratio of the stimulus display, (b) setting the center of gaze when the subject is believed to be gazing at the center, and (c) adjusting the gain until the calculated position of gaze (POG) point appears to match the non-central points of gaze. This effects the initial calibration mapping.	
<code>VPX_SendCommand ("FKey_cmd 9 { manualCalibration_Center }");</code>	

19.12.31 Initial Calibration Flip (Mirroring)

GUI:	<i>EyeSpace window > Advanced button</i> <i>Advanced window > Init: [x] Horizontal Flip [x] Vertical Flip</i>																																																																
CLI :	<code>calibration_InitWithHorizontalFlip BoolValue</code> <code>calibration_InitWithVerticalFlip BoolValue</code> <code>BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle</code>																																																																
Default	0.0																																																																
<p>This effects the initial calibration mapping only. It does <u>not</u> affect calibration presentation order. It is desirable for the system to startup with at least a good rough calibration, where the eye movements are at least in the correct directions. Mirrors and camera orientation can flip these, so the software allows the user to correct for this. Setting these correctly is especially important when using Manual Calibration. The flip options persist during Manual Calibration adjustments.</p>																																																																	
<table border="1" style="width: 100%; text-align: center;"> <thead> <tr> <th colspan="4">No Flip</th> <th colspan="4">Horizontal Flip</th> <th colspan="4">Vertical Flip</th> <th colspan="4">Horizontal & Vertical Flip</th> </tr> </thead> <tbody> <tr> <td>10</td><td>7</td><td>4</td><td>1</td> <td>1</td><td>4</td><td>7</td><td>10</td> <td>12</td><td>9</td><td>6</td><td>3</td> <td>3</td><td>6</td><td>9</td><td>12</td> </tr> <tr> <td>11</td><td>8</td><td>5</td><td>2</td> <td>2</td><td>5</td><td>8</td><td>11</td> <td>11</td><td>8</td><td>5</td><td>2</td> <td>2</td><td>5</td><td>8</td><td>11</td> </tr> <tr> <td>12</td><td>9</td><td>6</td><td>3</td> <td>3</td><td>6</td><td>9</td><td>12</td> <td>10</td><td>7</td><td>4</td><td>1</td> <td>1</td><td>4</td><td>7</td><td>10</td> </tr> </tbody> </table>		No Flip				Horizontal Flip				Vertical Flip				Horizontal & Vertical Flip				10	7	4	1	1	4	7	10	12	9	6	3	3	6	9	12	11	8	5	2	2	5	8	11	11	8	5	2	2	5	8	11	12	9	6	3	3	6	9	12	10	7	4	1	1	4	7	10
No Flip				Horizontal Flip				Vertical Flip				Horizontal & Vertical Flip																																																					
10	7	4	1	1	4	7	10	12	9	6	3	3	6	9	12																																																		
11	8	5	2	2	5	8	11	11	8	5	2	2	5	8	11																																																		
12	9	6	3	3	6	9	12	10	7	4	1	1	4	7	10																																																		
<p>See also: videoMirror</p>																																																																	
<pre>VPX_SendCommand ("calibration_InitWithHorizontalFlip TRUE");</pre>																																																																	

19.13 Controls: Criteria Controls

19.13.1 Specify amount of Smoothing

GUI: *Controls window > Criteria tab > Smoothing Points slider*

CLI : *smoothingPoints IntValue*

Default: **4**

The amount of smoothing to apply to the gaze position. The number specifies the number of previous sample points to use in the trailing average. A value of 4 makes attractive and useful real-time graphics. The value 1 indicates to use only the current point, i.e., no smoothing.

See also: [smoothingMethod](#) [velocityCriterion](#)

Modified: 2.8.2.44

`VPX_SendCommand ("smoothingPoints 3");`

19.13.2 Specify Smoothing Algorithm to Apply

GUI: *Controls window > Criteria tab > Smoothing Method pulldown list*

CLI : *smoothingMethod method*
method: SMA, EMA

Default: **SMA**

The user may choose between two smoothing algorithms: Simple Moving Average (SMA) and Exponential Moving Average (EMA).

The SMA method uniformly averages N pointsBack, i.e., all points having equal weight.

$$\text{SMA}(t) = [x(t) + x(t-1) + \dots + x(t-n)] / N ; \text{ where } n = (N-1)$$

The EMA method uses the following algorithm:

$$\text{EMA}(t) = (\text{currentValue} - \text{EMA}(t-1)) * K + \text{EMA}(t-1) ; \text{ where } K = 2 / (\text{pointsBack} + 1).$$

The number of pointsBack is adjusted by the *Smoothing slider* or the CLI: *smoothingPoints*.

Added: 2.8.2.44

`VPX_SendCommand ("smoothingMethod EMA");`



Arrington Research

19.13.3 Specify Velocity Threshold

GUI: *Controls window > Criteria tab > Saccade Velocity slider*

CLI : *velocityCriterion NormalizedValue*

velocityThreshold NormalizedValue // deprecated Form

NormalizedValue: floating point number in range 0.0 to 1.0

Default: **0.10**

The velocity level that is used to distinguish between saccades and fixations.

This value is displayed in the *PenPlot* window *TVelocity* plot as an upper level line with vertical arms going up to the top.

Note that the velocity magnitude, and consequently the required threshold, will be affected by the amount of smoothing applied.

See also: *smoothingPoints; driftCriterion, fixationMinimumSecondsCriterion; penPlot +TVelocity*

VPX_SendCommand ("velocityCriterion 0.8"); // Preferred

19.13.4 Specify amount of Drift Allowed

GUI: *Controls window > Criteria tab > Fixation Drift Allowed slider*

CLI : *driftCriterion NormalizedValue*

NormalizedValue: floating point number in range 0.0 to 1.0

Default: **0.03**

Specifies the absolute drift away from the fixation start point that is tolerated before a Drift classification is made. The units are in normalized stimulus window coordinates, just like gaze point. Without a Drift Criterion, the eyes can slowly change position and still be classified as a Fixation, because the velocity never exceeded the Saccade Velocity Criterion.

This value is displayed in the *PenPlot* window *Drift* plot as an upper level line with vertical arms going up to the top.

See also: *velocityCriterion, fixationMinimumSecondsCriterion; penPlot +DRIFT*

Changes: Default value changed in version 2.8.3 from 0.1 to 0.03

VPX_SendCommand ("driftCriterion 0.025");



Arrington Research

19.13.5 Specify Fixation Time Criterion

GUI: *Controls window > Criteria tab > Fixation Time Criterion*

CLI : **fixationMinimumSecondsCriterion seconds**

seconds: a floating point number in range 0.0 to 1.0

Default: **0.070 == that is 70 milliseconds**

Specifies the minimum fixation time required for a tentative fixation to be classified as a Fixation Event. This criterion level is displayed in the *PenPlot* window *Fixation Time* graphicsWell as a lower level line with vertical legs going down to the bottom.

See also: **velocityCriterion, driftCriterion, penPlot +FIXATION**

VPX_SendCommand ("fixationMinimumSecondsCriterion 0.080");

19.13.6 Specify Pupil Minimum AspectRatio Failure Criterion

GUI: *Controls window > Criteria tab > Pupil Aspect Criterion*

CLI : **pupilMinAspectCriterion NormalizedValue**

NormalizedValue: a floating point number in range 0.0 to 1.0

Default: **0.05**

Specifies the aspect ratio at which the data quality marker indicates that there is a problem. The default value is 0.05, so that any aspect ratio is accepted. Typically a useful value is about 0.8 Adjust the pupil aspect criterion for classification of blinks.

This value is displayed in the penplot window Pupil Aspect (Blinks) as a lower level line with vertical legs going down to the bottom. The pupil oval fit changes color from yellow to orange when criterion is violated.

Note: **pupilAspectCriterion** is deprecated in favor of: **pupilMinAspectCriterion**

See also: **penPlot +Aspect** and **pupilMaxAspectCriterion**

VPX_SendCommand ("pupilMinAspectCriterion 0.8");

19.13.7 Specify Pupil Maximum AspectRatio for PupilAngle to Change.

GUI:	<i>none</i>
CLI :	pupilMaxAspectCriterion <i>NormalizedValue</i> <i>NormalizedValue</i> : a floating point number in range 0.0 to 1.0
Default:	0.95
Specifies the aspect ratio above which the pupilAngle value will not change (hysteresis). This reduces noise when the pupil is essentially circular.	
This value is displayed in the penplot window Pupil Aspect (Blinks) as a upper level line with vertical legs going up to the top. The pupil oval fit changes color from yellow to orange when criterion is violated.	
See also: penPlot +Aspect and pupilMinAspectCriterion	
<code>VPX_SendCommand ("pupilMaxAspectCriterion 0.90");</code>	

19.13.8 Specify Pupil Min & Max Size Failure Criterion

GUI:	<i>Controls window > Criteria tab > Maximum Pupil Width slider</i>
CLI :	pupilMinWidthCriterion <i>NormalizedValue</i> pupilMaxWidthCriterion <i>NormalizedValue</i> <i>NormalizedValue</i> : a floating point number in range 0.0 to 1.0
Default:	Max = 0.75, Min = 0.00
Specifies the pupil width at which the data quality marker indicates that there is a problem.	
This value is displayed in the penplot window Pupil Major-axis or Pupil Width (depending on the pupil segmentation method). The Max value is the upper level line with vertical arms going up to the top; the Min value is the lower level line with vertical legs going down to the bottom.	
See also: penPlot +Width	
<code>VPX_SendCommand ("pupilMaxWidthCriterion 0.95");</code> <code>VPX_SendCommand ("pupilMinWidthCriterion 0.35");</code>	

19.14 Region of Interest (ROI)

19.14.1 Define an ROI Box

GUI:	<i>-none-</i>
CLI :	setROI_RealRect Index L T R B Shape <i>Index:</i> Integer value indicating the ROI to specify. <i>L T R B:</i> Normalized floating point values for the four sides. <i>Shape:</i> any of: { Rectangle, Ellipse }
Default:	One box at the center and eight iso-eccentric boxes.
Defines a Region of Interest (ROI) (aka window discriminator box). The first value <n> specifies which ROI to adjust. The next four values are the normalized (0.0 to 1.0) coordinates of the bounding rectangle: Left, Top, Right, Bottom	
See also: setROI_AllOff setROI_isoEccentric setROI_Shape	
<pre>VPX_SendCommand("setROI_RealRect 5 0.1 0.1 0.9 0.9 Ellipse"); // Sets ROI #5 with 10% margins VPX_GetROI_RealRect(n,&rr);</pre>	

19.14.2 Set ROI Name

GUI:	<i>-none-</i>
CLI :	setROI_Name NameString <i>NameString:</i> quoted text string
Sets a name string for an ROI.	
Changes: Version 2.9.3.121 the ROI_Name now appears in the <i>GazeSpace</i> window when the ROI is selected, and in the <i>Events</i> window when an ROI event is reported.	
<pre>VPX_SendCommand("setROI_Name 2 'Try Again' ");</pre>	

19.14.3 Draw IsoEccentric

GUI:	<i>-none-</i>
CLI :	setROI_isоЕccentric NumberOfROI SizeOfROI, Radius, AspectRatio <i>NumberOfBoxes :</i> Integer number of ROI in range {1 <= n <99} <i>SizeOfROI, Radius:</i> Normalized value in range {0.0 < s < 1.0} and {0.0 < r <0.5}. <i>AspectRatio :</i> Floating point ratio of the Width/Height of display in range {0.0 < a < 9.0}
Default:	NumberOfROI=8, SizeOfRoi=0.12, Radius=0.3, AspectRatio=1.333 = (4/3)
Convience method for creating a distribution of ROI, such that ROI#1 is in the center and ROI#2 through (NumberOfROI+1) are evenly spaced clockwise around it in an isoeccentric distribution, last one at top. Version 2.9.3.126 extended this method to include optional arguments SizeOfROI, etc.	



Arrington Research

```
VPX_SendCommand( "setROI_isoEccentric 8 0.12 0.25 1.33" );
```

19.14.4 Remove all ROI Boxes

GUI: *-none-*

CLI : **setROI_AllOff**

Default: *-none-*

Removes all ROI boxes.

```
VPX_SendCommand( "setROI_AllOff" );
```

19.14.5 Select a Specific ROI

GUI: *Controls window > Regions tab > Region of Interest slider*

Right mouse-click inside of ROI in the GazeSpace window

CLI : **setROI_Selection index**

index : integer number of the ROI to be selected. i.e. 1 or 2 or 3 etc. Up to the max number of ROI boxes enabled.

Default: *-none-*

Select ROI number **N**. The selected ROI is shown in red when the ROI overlay graphics are shown.

This can be used to highlight one particular ROI.

See also: **setROI_Lock**; **GazeSpace_MouseAction**

```
VPX_SendCommand( "setROI_Selection 9" );
```

19.14.6 Select the Next ROI Box

GUI: *Controls window > Regions tab > Region of Interest slider*

CLI : **setROI_selectNext**

Select the next ROI number (current + 1). The selected ROI is drawn in red when the ROI overlay graphics is shown. This can be used to highlight an area.

HINT: A mouse wheel is extremely helpful for moving between selections of the region slider.

```
VPX_SendCommand( "setROI_selectNext" );
```

```
VPX_SendCommand( "FKey_cmd 9 { setROI_selectNext } " );
```



Arrington Research

19.14.7 Lock ROI Settings

GUI: *Controls window > Regions tab > Region of Interest slider (Locked when slid left)*

CLI : **setROI_Lock**

Deselects all ROI, i.e. no ROI is selected.

See also: [setROI_Selection](#)

`VPX_SendCommand("setROI_Lock");`

19.14.8 Associate ROI with a Particular Stimulus Image

GUI: *File > Images > Save Image ROI* *Alt-Shift-I*

CLI : **savelImageROI**

savelImageROI *differentFileName*

This command saves a special *Settings* file in the *Images* folder containing the positions of the currently specified regions of interest (ROI). It provides an easy way to associate an image and the ROI needed for that image. Every time an image is loaded, e.g. *~/Images/MyImage01.bmp*, *ViewPoint* also looks in the same folder for the file *~/Images/MyImage01.bmp_ROI.txt* that contains ROI specifications. These ImageROI files are really just special small *Settings* files that are closely associated with the image file. Though designed for ROI, they could also be edited, for example, to play a sound that is associated with the image.

`VPX_SendCommand("savelImageROI");`

19.14.9 Set ROI Shape

GUI: *Controls window > Regions tab > Region of Interest slider (Locked when slid left)*

CLI : **setROI_Shape** *Index Shape*

setROI_Shape *Shape*

Index: Integer value indicating the ROI to specify.

Shape: any of: { Rectangle, Ellipse }

If an ROI index is specified and valid, this sets that ROI to the specified shape; otherwise, if ROI index is not specified or is negative, this sets ALL the ROI to that shape.

See also: [setROI_RealRect](#)

`VPX_SendCommand("setROI_Shape 1 Ellipse");`

19.15 PenPlot Controls

Many raw, calculated and derived data can be viewed in real-time in the *PenPlot* window.

19.15.1 ViewPoint PenPlot names and their meanings

Table 15. PenPlot names and Meanings	
CLI: penPlot option	Meaning
XGaze, YGaze	Position relative to the Stimulus or <i>SceneCamera</i>
XAngle, YAngle	Angle derived from Screen size and distance measurements. Data is only valid after accurate <i>GeometryGrid</i> measurements are set.
XPupil, YPupil	Raw <i>EyeSpace</i> pupil position as a function of time
Tvelocity	Total Velocity Qualitative in (2D) <i>ViewPoint</i> , accurate in Degrees/Second in 3D <i>ViewPoint</i> & <i>3DWorkSpace</i> Graphically shows the <i>Saccade Velocity</i> threshold criterion as a horizontal line.
Pupil Major Axis (Width)	Pupil Major-axis in units normalized to the width of the <i>EyeSpace</i> window Graphically shows the <i>Maximum Pupil Width</i> and the <i>Minimum Pupil Width</i> threshold criteria as horizontal lines.
Height	Pupil Minor-axis in units normalized to the width of the <i>EyeSpace</i> window
Aspect	Pupil Aspect Ratio that is calculated as minor-axis divided by major-axis Dimensionless ratio that is always between 0.0 and 1.0 Graphically shows <i>Pupil Aspect Criterion</i> threshold for blink detection as a horizontal line.
Diameter	Pupil Major-axis scaled to millimeters. Data is only valid after accurate pupil calibration is set, see 9.3
Vergence	Calculated vergence angle of eye rays. IPD must be set..
GazePoint3D	Calculated gaze point (x,y,z) in cm. Must be separately calibrated.
Disparity	Separation on the screen of EyeA and EyeB gazePoints. Available only with binocular option
Torsion	Amount of eye rotation about the z-axis. (Torsion option only)
Cyclovergence	The difference between the torsion of the two eyes. (Torsion option only)
Drift	How far from the initial fixation point the current gazePoint is. Graphically shows <i>Fixation Drift Allowed</i> threshold as a horizontal line.
Fixation	The integrated fixation time.
Events	Classified EyeMovement Events
ROI	Which Regions Of Interest are hit. Only the first 8 are displayed in the penPlot.
FPS	Frames Per Second
Timing	The reciprocal of Frames Per Second (FPS). E.g. 16.67 for 60 Hz systems
Processing	Time difference between the videoTime and the storeTime <code>VPX_GetDateTime2(eyn, &videoTime); VPX_GetStoreTime2(eyn, &storeTime);</code> <code>processingTime = (storeTime - videoTime) * SecondsToMilliseconds;</code>
Quality	Data quality code
Seconds	Seconds and Markers
cpuLoad	Similar to the Windows Task Manager Note: you can specify the amount of smoothing to use for the display with the following command: cpuLoad_SetSmoothing percentPreviousValue Sets the amount of smoothing, must be ≥ 0.0 and < 1.0 Lower values (e.g. 0.1) are less smoothing, higher values (e.g. 0.9) are more smoothing. The CPU usage varies wildly from sample to sample, so pretty much smoothing is required.

19.15.2 3DViewPoint & 3DWorkSpace PenPlots

3DViewPoint and *3DWorkSpace* add additional PenPlots as well as changing some of the existing PenPlot names. Below are the PenPlot names matched with their CLI command equivalent name.

Table 16. PenPlot Names and Labels

CLI: penPlot option specifier (precede with + or -)	Label in PenPlot window when in 3D mode
ROI	3D ROI
Panel	3D Panel
Xgaze	3D Panel X
Ygaze	3D Panel Y
Xangle	3D Azimuth
Yangle	3D Elevation
Xvelocity	3D Azimuth Vel
Yvelocity	3D Elevation Vel
Tvelocity	3D Angular Vel
Vergence	3D Vergence
GazePoint3D	3D GazePoint
HeadPosition	Head Position (x,y,z; cm)
HeadAngle	Head Angle (y,r,p, deg) i.e. yaw, roll, pitch

19.15.3 PenPlot Dump Names

GUI: *-none-*

CLI : **penPlot_dumpNames**

Default:

Prints a list of each **penPlotName** in the *History* window.

`VPX_SendCommand("penPlot_dumpNames");`

19.15.4 Specify Which PenPlot Traces to Display

GUI: **PenPlots >**

CLI : **penPlot +penPlotName**
penPlot -penPlotName

penPlotName: use CLI **penPlot_dumpNames** to obtain a current list.

Default: **+Xgaze +Ygaze +Tvelocity +Width +Height +Aspect +Drift +Events +ROI +FPS**
UserData1, UserData2, UserData3, UserData4, UserData5

Specifies which penPlots to display in the *PenPlot* window. To include a plot precede its name string with a “+”, to exclude it precede its name with “-”. Be careful that there is no white space between + or - and the name. Separate multiple strings arguments on the same line with a space, as in the example below.

Use **+All** to show all penPlots, use **-All** to remove all penPlots.

The CLI command **penPlot_dumpNames** will list each **penPlotName** in the *History* window. This is useful for spelling them correctly. Some of the more common are described here.

`VPX_SendCommand("penPlot +Xgaze +Ygaze -Tvelocity +Width +Aspect -Timing");`



Arrington Research

19.15.5 PenPlot Background Color

GUI: *-none-*

CLI : **penPlot_BackgroundColor intRed intGreen intBlue**
 intRed 0-255 *intGreen* 0-255 *intBlue* 0-255

Default: 160 160 160

Specifies the color that fills the penPlot GraphicsWell rectangle background. The *PenPlot Limen* has its own fill color.

See also: [penPlot_LimenFillColor](#)

`VPX_SendCommand("penplot_BackgroundColor 115 100 235");`

19.15.6 PenPlot Limen Fill Color

GUI: *-none-*

CLI : **penPlot_LimenFillColor intRed intGreen intBlue**
 intRed 0-255 *intGreen* 0-255 *intBlue* 0-255

Default: 144 144 144

Specifies the color that fills the rectangle within the criterion lines (threshold, limen) in the penPlots. The default value is slightly darker gray than the default [penPlot_BackgroundColor](#). This is different from the colors of the upper (red) and lower (yellow) criterion lines themselves.

See also: [penPlot_BackgroundColor](#)

`VPX_SendCommand("penplot_LimenFillColor 115 100 235");`

19.15.7 Specify Speed of PenPlot Scrolling

GUI: *PenPlots > Speed > { ¼x, ½x, 1x, 2x, 4x, 8x, 16x }*

CLI : **penPlot_Speed xInc**
 xInc: float in range {0.25 to 16.0}

Default: 1

Specifies the number of pixels to increment the x (time) axis in the penPlot trace lines.

Sometimes it is desirable to increase the speed of the penPlot scrolling, so that details, markers, and classifications can be more easily distinguished.

Modified: 2.8.2.51, 2.8.3.445

`VPX_SendCommand("penPlot_speed 6");`



Arrington Research

19.15.8 Specify Size of PenPlot Lines

GUI: *PenPlots > PenSize > { 1, 2, 3, 4 }*

CLI : *penPlot_PenSize size*
size: integer in range {1 to 4}

Default: **1**

Specifies the thickness of the penPlot trace lines.

Modified: 2.8.3.445

`VPX_SendCommand("penPlot_penSize 4");`

19.15.9 Specify Range of PenPlot Values

GUI: *PenPlot window: Right mouse click in a lineGraph well:*
popup menu allows : Zoom In / Zoom Out / UnZoom

CLI : *penPlot_Range penPlotName lowerValue upperValue*
penPlotName: Use CLI penPlot_dumpNames to obtain a current list.
lowerValue, upperValue: float

Default: *varies depending on penPlotName*

Specifies the lower and upper plot range values for a particular penPlot graph.

The GUI *Zoom In / Zoom Out* adjust the range and also centers the range about the latest data point.

See also: *penPlot_dumpNames*

Added: 2.8.2.51

`VPX_SendCommand("penPlot_Range xgaze 0.2 0.6");`

19.15.10 Restart the PenPlot

GUI: *-none-*

CLI : **penPlot_restart**

Default: **No**

Erases the penPlot lines and restarts the lines at the left.

`VPX_SendCommand("penplot_restart");`

19.15.11 Specify the Behavior of the PenPlot after resuming from a VideoFreeze

GUI:	<i>-none-</i>
CLI :	penPlot_restartAfterFreeze BoolValue <i>BoolValue:</i> Yes, No, True, False, On, Off, 1, 0, Toggle
Default:	No
Specifies the behavior of the penPlot after resuming from a video freeze. Specifies whether or not the pen will continue going from where it is, or if the penplot window will be erased and the pen will reset and start afresh from the left.	
<code>VPX_SendCommand("penplot_restartafterfreeze Yes");</code>	

19.16 Graphics Controls

19.16.1 Specify the Eye Color for GUI graphics

GUI:	<i>Controls window > Display tab > Eye A or Eye B button</i>
CLI :	gazeColor intRed intGreen intBlue <i>intRed</i> 0-255 <i>intGreen</i> 0-255 <i>intBlue</i> 0-255
Default:	EyeA:gazeColor 100 255 0; EyeB:gazeColor :0 255 255 // limeGreen & cyan
Specifies the red, green and blue components (0 to 255) of the penPlot lines and the POG in the <i>GazeSpace</i> window and the <i>Stimulus</i> window. E.g., 255 255 255 is white and 100 100 255 is a sky blue. Note: This will also change the color of the Calibration Stimulus Points associated with that eye.	
See also: gazePoint_Transparency	
Do not use the deprecated commands: penColorA penColorB	
<code>VPX_SendCommand("EyeA:gazeColor 115 100 235"); // Purple</code>	
<code>VPX_SendCommand("EyeB:gazeColor 0 255 0"); // Green</code>	



Arrington Research

19.16.2 GazePoint Transparency

GUI: *-none-*

CLI : *gazePoint_Transparency NormalizedValue*

NormalizedValue: a floating point number in range 0.1 to 1.0

Specifies the transparency of the dots in the *GazeSpace* window. A value of 1.0 makes the dot opaque (no transparency); a value of 0.1 makes the dot

Specify the type

so transparent that it is barely visible.

See also: *gazeColor* and *gazeCursor_Transparency* (that takes a non-normalized argument 0 to 255)

VPX_SendCommand("gazePoint_Transparency 0.9");

19.16.3 Graphics Options for *GazeSpace* & *Stimulus* windows and *SceneMovie*

GUI: *Controls window > Display tab > Checkbox matrix*

CLI : *gazeSpaceGraphicsOptions +graphicsOptions*

stimulusGraphicsOptions +graphicsOptions

sceneMovieGraphicsOptions +graphicsOptions

graphicsOptions:

+ROI +POG +Path +Fix +Size +Grid +Cal +Raw +Image +Time +Markers

Defaults

GazeSpace +ROI +POG +Cal +Image +Time

Stimulus +POG +Image

SceneMovie +POG +Path +Time

Specifies which overlay graphics to display in the *GazeSpace* window, *Stimulus* window, or *SceneMovie*.

Use *-graphicsOption* to exclude an option, or use *+graphicsOption* to include an option.

Note: The +/- must be next to the key word, i.e., NOT separated from it by a space.

Separate multiple arguments by spaces, as in the example here below.

See also: *stimulusGraphicsOptions*; *sceneMovieGraphicsOptions*

Note that either *gazeSpaceGraphicsOptions* or *gazeGraphicsOptions* may be used.

Note that Markers are not currently painted into the SceneMovie.

VPX_SendCommand("gazeGraphicsOptions +POG -Raw -ROI");



Arrington Research

19.16.4 Erase Data Displays in the *GazeSpace* and *Stimulus* Windows

GUI: *Controls window > Display tab > Erase Displays button*

CLI : **eraseDisplays**

Clears the previous drawn data from the *GazeSpace* and *Stimulus* windows.

See also: **timedErase_autoErase;** **timedErase_delaySeconds**

`VPX_SendCommand("eraseDisplays");`

19.16.5 Automatically Erase Display Windows

GUI: **-none-**

CLI : **timedErase_autoErase BoolValue**
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: **Off**

Automatically erases the *Stimulus* and *GazeSpace* windows after timed delay is reached.

See also: **timedErase_delaySeconds;** **eraseDisplays**

`VPX_SendCommand("timedErase_autoErase On");`

`VPX_SendCommand("timedErase_ delaySeconds 6.5");`

19.16.6 Specify Time Delay for Auto Erase

GUI: **-none-**

CLI : **timedErase_delaySeconds seconds**
seconds : floating point number

Default: **3.5**

Specifies the time delay for automatically erasing the *Stimulus* and *GazeSpace* windows.

See also: **timedErase_autoErase;** **eraseDisplays**

`VPX_SendCommand("timedErase_autoErase On");`

`VPX_SendCommand("timedErase_ delaySeconds 6.5");`

19.17 Stimulus Window Controls

19.17.1 Specify Stimulus Source	
GUI:	<i>Stimuli > View Source ></i> <i>{ StimulusWindow, SceneCamera, Interactive Computer Display ... }</i>
CLI :	<i>viewSource options</i> <i>options: StimulusWindow, SceneCamera, 1, 2, 3</i>
Default:	StimulusWindow
<p>Specifies the type of stimulus the subject will be viewing.</p> <p>StimulusWindow : a typical psychophysical bench experiment with the head fixed, or an HMD.</p> <p>SceneCamera : subject is looking at the real world with the HeadMounted <i>SceneCamera</i> recording what the subject is viewing.</p> <p>1, 2, etc. : typically for usability experiments; the subject is interacting with a program or web browser on the computer screen; snapshots of the computer screen images are saved in a movie. The number specifies which display monitor to use.</p> <p>See also: calibration_PointLocationMethod</p>	
<pre>VPX_SendCommand("viewSource SceneCamera");</pre>	



Arrington Research

19.17.2 Specify Custom Stimulus Window Size and Position

GUI:	<i>1st) select style: Stimuli > Stimulus Window Properties > Normal Adjustable Window</i> <i>2nd) resize the window by dragging its border</i> <i>3rd) change style: Stimuli > Stimulus Window Properties > Custom Static Position</i>
CLI :	stimWind_CustomStatic L T R B <i>L T R B</i> : integer pixel values based on the display space.

Specifies a custom position and size of the *Stimulus* window. Where L,T,R,B correspond to the left, top, right and bottom physical dimensions across all monitors. The following example assumes that we have a secondary display placed in the virtual desktop with the tops of the two displays at the same level, namely zero. Further, the primary monitor is 1280 x 1024 and the secondary is 1024 x 768, such that the top left origin of the primary monitor is at (0,0), the top left origin of the secondary monitor is at (1280,0), and the extreme diagonal bottom right point is at (1280+1024,768) or (2304,768)

stimWind_CustomStatic 1280 0 2304 768.

Sets the custom static *Stimulus* window in the position specified. Displays the *Stimulus* window as a static window (no borders or title bar). The window will be always in front. This is useful if the graphics card does not show a second display. Custom was primarily developed to help users who have dual monitor display card that do not correctly tell the Windows OS that there are two devices; these are now rare. This does not causes the *Stimulus* window to be shown if it's hidden.

See also: **setStimulusDisplay Custom;** **setWindow Stimulus Show;** **stimulus_ImageHidden**

```
VPX_SendCommand( "stimWind_CustomStatic 1280 0 2304 768" );  
VPX_SendCommand( "setWindow Stimulus Show" );
```



Arrington Research

19.17.3 Automatically Show the Stimulus Window upon Calibrate

GUI: *Stimuli > Stimulus Window Properties > AutoShow upon Calibrate*

CLI : *stimwind_AutoShowOnCalibrate BoolValue*
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: **Yes**

ViewPoint has been designed to make calibration easy and fast, even when there is only one computer display, and when the experimenter wants to do self-testing.

If the *AutoShowOnCalibrate* feature is active *ViewPoint* will automatically show the stimulus window whenever a calibration start procedure begins.

See also: [setStimulusDisplay 1](#)

`VPX_SendCommand("stimwind_autoshowncalibrate No");`

19.17.4 Show SceneVideo in *Stimulus* window

GUI: *Controls window > Scene tab > Show SceneVideo in Stimulus window checkbox*

CLI : *ShowSceneVideoInStimulusWindow BoolValue*
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: **Off**

By default the video from the *SceneCamera* is displayed only in the *GazeSpace (SceneCamera)* window. This causes the *SceneVideo* to be displayed in the *Stimulus* window as well.

Version 1.9.3.123 changed this from GUI “*Stimulus Scene Video*” and CLI: *stimulus_SceneVideo*, both now deprecated, because the language was unclear.

`VPX_SendCommand("ShowSceneVideoInStimulusWindow Toggle");`

19.17.5 Specify How and Where to Show Stimulus Window

GUI:	<i>Stimuli > Stimulus Window Properties > Normal Adjustable</i> <i>Stimuli > Stimulus Window Properties > Custom Static</i> <i>Stimuli > Stimulus Window Properties > Full Screen Monitor 1 (Primary)</i> <i>Stimuli > Stimulus Window Properties > Full Screen Monitor 2</i> <i>Stimuli > Stimulus Window Properties > Full Screen Monitor 3</i> <i>etc.</i>
CLI :	setStimulusDisplay displayOption <small>stimWind_FullDisplay displayOption // Deprecated</small> <i>displayOption</i> : Adjustable, Custom, 1, Primary, 2, Secondary
Default:	1
	Specifies where and how to display the stimulus window. Arguments Primary , 1 , Secondary , 2 , etc. specify that the stimulus window will be displayed full screen on these monitors. Adjustable indicates that the stimulus window is to be displayed as a regular adjustable-size floating window. Custom was primarily developed to help users who have dual monitor display card that do not correctly tell the Windows OS that there are two devices; these are now rare.
	Note 1: This command ONLY specifies the selection of the device on which the <i>Stimulus</i> window will be displayed (or is displayed, if the stimulus window is currently showing), it does NOT cause the stimulus window to be shown. There are advantages to separating these instructions for (i) selecting the device and for (ii) showing / hiding the window, not the least of which is isomorphism to the conceptual layout of the GUI menu item groupings.
	Note 2: In previous versions, prior to 2.8.3, the argument 0 corresponded to the instruction to Hide the stimulus window, or in some previous versions, to toggle Show / Hide. Argument 0 should no longer be used and its effect may be unpredictable.
See also:	setWindow STIMULUS SHOW setWindow STIMULUS HIDE stimWind_CustomStatic stimWind_AutoShowOnCalibrate
Modified:	2.8.2.52
	<pre>VPX_SendCommand("setStimulusDisplay Seconday");</pre>

19.18 Window Related Controls

19.18.1 Size Window	
GUI:	<i>Resize window with mouse.</i>
CLI :	sizeWindow <i>windowNameString W H</i> <i>windowNameString</i> : Main, EyeA, EyeB, EyeSpace, Controls, Status, GazeSpace, PenPlot, History, Events, Geometry, Observer, etc. (not filtered) <i>W H</i> : two integers describing the width and height of the window content rectangle.
Sizes / resizes the specified window to the specified width and height. The left-top corner position remains the same; the right-bottom corner is adjusted to satisfy the new size.	
<i>See also:</i> setWindow moveWindow settingsFile_saveWindowLayout	
VPX_SendCommand("sizeWindow GazeSpace 640 480");	

19.18.2 Move Window, or Move & Resize Window	
GUI:	<i>Drag window with mouse, resize window with mouse.</i>
CLI :	moveWindow <i>windowNameString L T</i> moveWindow <i>windowNameString L T W H</i> <i>windowNameString</i> : Main, EyeCamera, EyeA, EyeB, EyeSpace, Controls, Status, GazeSpace, PenPlot, History, Events, Geometry, etc. <i>L T W H</i> : the first two integers specify the left, top corner of the window; the second two integers are optional and specify the width and height of the content area of the window.
Moves the specified window so the left-top corner is at the specified location. Optionally the window can be resized at the same time by additionally specifying the width and height.	
If the window is Free (see setWindow Free), the location can be outside the <i>ViewPoint</i> parent window, and can be on other monitors if multiple monitors are present.	
<i>See also:</i> setWindow sizeWindow	
VPX_SendCommand("setWindow GazeSpace Free; moveWindow GazeSpace -900 50");	



19.18.3 Specify ViewPoint Window State, or Windows Layout

GUI:	<i>Windows > { WindowName } (Hold the SHIFT-KEY to Toggle showing the window)</i> <i>Windows > Arrange > { Startup Layout, Larger Layout, Cascade }</i> <i>Window minimize and maximize title bar boxes.</i>
CLI :	<code>setWindow windowNameString windowState</code> <code>setWindow windowLayout</code> <i>windowNameString:</i> Main, EyeA, EyeB, EyeSpace, Controls, Status, GazeSpace, PenPlot, History, KeyPad, Events, Geometry, CalibrationImage, Histogram, About, Info, TorsionA, TorsionB, etc. <i>windowState:</i> Show, Hide, Maximize, Minimize, Restore, Free, Child, FreeToggle <i>windowLayout:</i> Startup, Larger, Cascade <code>layout layoutName</code> <i>windowNameString:</i> 1920x1080, 1280x1024, 1024x768

Users can load program defined layouts, e.g. CLI: `layout 1280x1024`, or they can specify and load their own layouts or those pre-defined in files in: ...\\ViewPoint\\Settings\\Layouts\\ or select menu item: *Windows > Arrange > Larger Layout SXGA (1280x1024)*.

Allows the user to easily control the state of all *ViewPoint EyeTracker* windows.

`setWindow Startup` : equivalent to: *Windows > Arrange > Startup Layout (1024x768)*

`setWindow Larger` : equivalent to: *Windows > Arrange > Larger Layout (1280x1024)*

`setWindow Cascade` : equivalent to: *Windows > Arrange > Cascade*

Note: `setWindow Stimulus Hide` is different from the command `stimulusGraphicsOptions -Image`; the latter suppresses showing the bitmap image inside the *Stimulus* window.

Use the `Free` option to allow the specified window to be moved outside the *ViewPoint* parent window; use the `Child` option to place it back inside the *ViewPoint* parent window; or `FreeToggle` option.

The `Maximize` option make the window full screen, and the `Minimize` option hides the window except for an icon at the bottom of the screen, just as the buttons in the window title bar do; the `Restore` option returns the window to the previous state.

See also: `layout, moveWindow sizeWindow setStimulusDisplay stimWind_AutoShowOnCalibrate`

Deprecated: `stimWind_Hide`

Version 2.9.3.124 added `FreeToggle`

```
VPX_SendCommand( "setWindow Stimulus Show" );
VPX_SendCommand( "setWindow GazeSpace Free" );
VPX_SendCommand( "setWindow PenPlot Maximize" );
```



Arrington Research

19.18.4 History Window Options

GUI: *-none-*

CLI : *historyOptions +/-optionString*
optionString: +timeStamp

Allows greater control of the information that appears in the *History* window.

Use *-optionString* to exclude an option, or use *+optionString* to include an option.

Note: The *+/* must be next to the key word, i.e., NOT separated from it by a space.

```
VPX_SendCommand( "historyOptions -timeStamp" );
// Remove timestamp from lines printed in the History window.
```

19.18.5 History User Text

GUI: *-none-*

CLI : *historyReport "quotedString"*
say 'quotedString'

The user may send text strings to the *History* window using these commands. This is useful in *Settings* files, to describe what has been loaded. It is also an easy way for layered applications to report information.

Version 2.9.2.2 strings can be either single or double quoted and may have one level of nesting (single quotes inside double quotes, or double quotes inside single quotes) is allowed.

For help on Quoting Strings see section [18.9](#).

```
VPX_SendCommand( "say 'Hi there.' " );
VPX_SendCommand( "HistoryReport 'Hi there.' " );
```

19.18.6 Clear History Window

GUI: *Windows > Clear History*

History window ControlMenu > Clear Window (pull down menu item)

CLI : *clearHistory*

Clears the contents of the *History* window.

See also: *clearEvents*

```
VPX_SendCommand( "clearHistory" );
```



Arrington Research

19.18.7 Clear Events Window

GUI:	<i>Windows > Clear Events</i>
	<i>Events window ControlMenu > Clear Window (pull down menu item)</i>
CLI :	<code>clearEvents</code>
Clears the contents of the <i>Events</i> window.	
See also: clearHistory	
<code>VPX_SendCommand("clearEvents");</code>	

19.18.8 GazeSpace MouseAction

GUI:	<i>Controls window > Regions tab > radio buttons</i>
CLI :	<i>GazeSpace_MouseAction selection</i> <i>selection : None, Content, CalibRegion, GazeNudge, Simulation, ROI</i>
Specifies what the mouse will do in the <i>GazeSpace</i> window.	
<code>VPX_SendCommand("GazeSpace_MouseAction Content");</code>	

19.19 Settings File Commands

19.19.1 Load Settings File

GUI:	<i>File > Settings > Load Settings ...</i> ^L
CLI :	<code>settingsFile_Load fileName</code> <code>settingsFile_Load folderName</code> <i>fileName</i> : Name of the <i>Settings</i> file including file extension <i>folderName</i> : Name of a folder containing <i>Settings</i> files to be loaded.
Loads a <i>Settings</i> file of the specified <i>fileName</i> . Recursion is disallowed.	
Nesting depth is limited to 9.	
The <i>fileName</i> should include the extension.	
If a <i>folderName</i> is used instead of a <i>fileName</i> , <u>all top level files in that folder</u> (but not sub-folders) will be loaded.	
For help on Quoting Strings see section 18.9 .	
See also: <code>settingsFile_SaveWindowLayout layout</code>	
<code>VPX_SendCommand("settingsFile_Load 'researchsettings.txt' ");</code>	



Arrington Research

19.19.2 Edit Settings File

GUI: *File > Settings > Edit Settings File ...*

CLI : **settingsFile_EditDialog**

Raises a file browser window so the user can quickly view and edit files in the *Settings* folder.

```
VPX_SendCommand( "settingsFile_EditDialog" );
```

19.19.3 Verbose CLI Parsing and loading of Settings Files

GUI: *File > Settings > Verbose Loading*

CLI : **Verbose +Setting +Parsing**
 settingsFile_Verbose BoolValue // deprecated

Default: **No**

Specifies whether to report verbose feedback when parsing CLI commands and loading *Settings* Files.

See also: **verbose +/-Option** see: [19.28.1](#)

```
VPX_SendCommand( "verbose +Setting +Parsing" );
```

19.19.4 Save Settings e.g. Calibrations etc.

GUI: *File > Settings > Save Settings ...* **Alt-Shift-S**

CLI : **settingsFile_Save Filename**

Filename: Name of the Settings file to be saved with no extension

Saves a *Settings* file with the specified *fileName*. The file extension '.txt' will be added, so no extension should be specified in the *fileName* string.

See also: **settingsFile_SaveWindowLayout settingsFile_Load**

```
VPX_SendComand( "settingsFile_save 'goodSettings' " ); // No extension needed  
// Creates: 'goodSettings.txt'
```



Arrington Research

19.19.5 Save Window Layout Settings

GUI: *File > Settings > Save Window Layout*

settingsFile_SaveWindowLayout filename

CLI :
filename: Name of the Settings file to be saved.

The window layout information can be saved in a special *Settings* file. This includes the size and position of the window. It is not saved as part of the standard Save Settings operation. The file extension ‘.txt’ will be added, so no extension should be specified in the fileName string.

Note: If you want a Free window full screen on a second monitor, you must move it and drag it full screen; do not MAXIMIZE the window. If the window is saved with the MAXIMIZE parameter then it will most probably maximize over the *ViewPoint* main window.

See also: **settingsFile_Save settingsFile_Load**

```
VPX_SendCommand( "settingsFile_SaveWindowLayout 'myWindowlayout_large' " );
```

```
// File will contain many lines including some that look like this:
```

```
...
setWindow CHILD EyeA
setWindow SHOW EyeA
moveWindow EyeA 1 1 350 265
setWindow FREE PenPlot
setWindow SHOW PenPlot
moveWindow PenPlot -1049 -391 1051 1680
setWindow HIDE Geometry
setWindow HIDE KeyPad
...
```

19.20 SettingsFileList (DEPRECATED. Please use the *StateEngine*)

Note that the *SettingsFileList* mechanism is deprecated in favor of the *StateEngine*.

19.20.1 Initialize Settings File List

GUI: -none-

CLI : **settingsFileList_Init**

Initializes the list for *Settings* file, making it ready for new *Settings* to be entered.

Note that the *SettingsFileList* mechanism is deprecated in favor of the *StateEngine*.

```
VPX_SendCommand( "settingsFileList_Init" );
```



Arrington Research

19.20.2 Next Settings File in List

GUI: File > Settings > SettingsFileList > Next Settings File F9 (default)

CLI : settingsFileList_Next

Executes the settings for the next *Settings* file in the *SettingsFileList*.

Note that the *SettingsFileList* mechanism is deprecated in favor of the *StateEngine*.

VPX_SendCommand("settingsFileList_Next");

19.20.3 Add Settings File to the List

GUI: -**none-**

CLI : settingsFileList_AddName *FileName*
FileName: Name of the *Settings* file to be added to the *Settings* file list.

Adds a *Settings* file name to the *Settings* file list.

Note that the *SettingsFileList* mechanism is deprecated in favor of the *StateEngine*.

VPX_SendCommand ("settingsFileList_AddName subject1.txt");

19.20.4 Restart Settings File List

GUI: File > Settings > SettingsFileList > Restart SettingsFileList

CLI : settingsFileList_Restart

Re-opens the *Settings* file list and re-applies the settings listed in the *Settings* files.

Note that the *SettingsFileList* mechanism is deprecated in favor of the *StateEngine*.

VPX_SendComand("settingsFileList_Restart");

19.20.5 Toggle Autosequencer ON / OFF

GUI: File > Settings > SettingsFileList > Auto-Sequencer F10 (default)

CLI : settingsFileList_AutoSequence *Bool/Value*
Bool/Value: Yes, No, True, False, On, Off, 1, 0, Toggle

Enables or disables *Settings* file list sequencer.

Note that the *SettingsFileList* mechanism is deprecated in favor of the *StateEngine*.

VPX_SendCommand("settingsFileList_AutoSequence ON");



Arrington Research

19.20.6 Specify Delay between Settings Files in List

GUI: -none-

CLI : settingsFileList_SequenceSeconds *TimeValue*
TimeValue: Amount of sequence seconds.

Creates a delay between loading *Settings* files from the *Settings-file-list*. Users may also specify sequence-seconds within the *Settings* files themselves rather than in a *Settings-file-list*.

Will create a delay in the execution of *Settings* files from the list for as long as specified. Also note that the sequence seconds can be set within a *Settings* file itself.

Note that the *SettingsFileList* mechanism is deprecated in favor of the *StateEngine*.

VPX_SendCommand("settingsFileList_SequenceSeconds 4.5");

19.20.7 Randomize Settings Files

GUI: -none-

CLI : settingsFileList_Randomize

Randomizes *Settings* files in the list.

Note that the *SettingsFileList* mechanism is deprecated in favor of the *StateEngine*.

VPX_SendCommand("settingsFileList_Randomize");

19.21 Torsion Commands

19.21.1 Start / Stop Torsion Calculations

GUI: *Torsion window, Start / Stop button* ^T

CLI : torsion_Calculation *BoolValue*
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Off

Starts and stops the torsion calculations. As a side effect, [*Start*] adds the *Torsion* lineGraph to the *PenPlot* window.

Note: In previous PC versions this command would open and close the torsion window as well, because in those previous versions, torsion would be calculated if and only if the torsion window was open.

See also: [setWindow Torsion Show](#)

VPX_SendCommand("torsion_Calculation On");



Arrington Research

19.21.2 Adjust Radius of Torsion SamplingArc

GUI:	<i>Torsion window, Radius slider</i>
CLI :	torsion_SampleRadius <i>FloatValue</i> <i>FloatValue</i> : normalized floating point number 0.01 - 0.99
Default:	0.30
Adjust the radius (the distance out from the center of the pupil) of the torsion SamplingArc. The arc should be adjusted such that: (a) there is good variation in the striations and marks of the iris, (b) the arc does not include any reflections, such as the glint, that do not move with the iris, (c) the arc does not extend beyond the <i>EyeCamera</i> window.	
<code>VPX_SendCommand("torsion_SampleRadius 0.75");</code>	

19.21.3 Adjust Start Point of Torsion SamplingArc

GUI:	<i>Torsion window, Angle slider</i>
CLI :	torsion_SampleAngle <i>FloatDegrees</i> <i>FloatDegrees</i> : floating point value 0.0 to 360.0 degrees
Default:	250
Adjusts the <u>starting point</u> of the torsion SamplingArc in degrees. Range is from 0.0 to 360.0 degrees. Zero degrees is at the 3 o'clock position, 90 degrees is at the 6 o'clock position. An interesting demonstration and validation can be obtained on a static image by first unchecking the <i>Auto-set after adjust check</i> box and then moving the Angle slider a few degrees. This moves the start point of the sample vector and so the autocorrelation shows a "torsional" rotation of the same number of degrees.	
<code>VPX_SendCommand("torsion_SampleAngle 166");</code>	

19.21.4 Adjust Length of Torsion SamplingArc

GUI:	<i>Torsion window, ArcDeg slider</i>
CLI :	torsion_SampleArc <i>IntValue</i> <i>IntValue</i> : integer number 30 - 360
Default:	120
Adjust the <u>arc length</u> of the torsion SamplingArc, increasing clockwise. The arc should be adjusted such that: (a) there is good variation in the striations and marks of the iris, (b) the arc does not include any reflections, such as the glint, that do not move with the iris, (c) the arc does not extend beyond the <i>EyeCamera</i> window.	



```
VPX_SendCommand( "torsion_SampleRadius 0.75" );
```

19.21.5 Autoset Torsion Template after Adjustments

GUI:	<i>Torsion window, Auto-Set after adjust checkbox</i>
CLI :	torsion_AutoSetAfterAdjust BoolValue <i>BoolValue</i> : Yes, No, True, False, On, Off, 1, 0, Toggle
Default:	On
If ON, a new autocorrelation template is set whenever adjustments are made to the radius, starting angle, or arc length.	
<pre>VPX_SendCommand("torsion_autosetafteradjust On");</pre>	

19.21.6 Display Real-Time Torsion Data

GUI:	<i>Torsion window, Real-time graphics checkbox</i>
CLI :	torsion_RealTimeGraphics BoolValue <i>BoolValue</i> : Yes, No, True, False, On, Off, 1, 0, Toggle
Default:	OFF for USB-220 systems, ON otherwise
Specifies whether to display real-time torsion data in the torsion window. If ON the window is updated without apparent delay. If OFF, the window is updated about every half second. This does not affect the real-time data stored in the data file, but does effect the CPU load.	
<pre>VPX_SendCommand("torsion_RealTimeGraphics On");</pre>	



Arrington Research

19.21.7 Adjust Torsion Measurement Range

GUI: *-none-*

CLI : **torsion_MeasureDegrees** *FloatDegrees*

Default: +/- 20.0

Adjusts the torsion measurement range.

Since the eye does not normally rotate about the line-of-sight more than about 20 degrees there is usually no need to perform the auto-correlation past this range, because increasing the range increases the CPU load unnecessarily. There are some situations in which this range needs to be increased, such as when the entire head is rotated. Depending upon the power of your computer, you may need to reduce the resolution of the auto-correlation via: **torsion_ResolutionDegrees**.

See also: **torsion_ResolutionDegrees**

`VPX_SendCommand("torsion_MeasureDegrees 9.0");`

19.21.8 Adjust Torsion Measurement Resolution

GUI: *-none-*

CLI : **torsion_ResolutionDegrees** *FloatDegrees*

FloatDegrees: floating point value between: 0.20 to 360.0

Default 0.5

Adjusts the default torsion measurement resolution. This minimum value is 0.20.

To limit CPU load, vary this inversely with **Torsion_MeasureDegrees**.

See for further discussion: **torsion_MeasureDegrees**

`VPX_SendCommand("torsion_ResolutionDegrees 0.80");`

19.21.9 Set Autocorrelation Template

GUI: *Torsion window, Set Template button*

CLI : **torsion_SetTemplate**

Sets or Re-sets autocorrelation template.

See: **torsion_AutoSetAfterAdjust**

`VPX_SendCommand("torsion_SetTemplate");`

19.22 Interface Settings Commands



19.22.1 GazeCursor On / Off

GUI: *Interface > CursorControl > GazeCursor* ^C

CLI : *gazeCursor_Used BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Off

Controls whether or not to display the *gazeCursor*, a transparent circle on the monitor where the subject is looking.

This *gazeCursor* is entirely separate from the *mousePointer* and will not move the *mousePointer*, unless you also select *Eye Moves Mouse* / *cursor_Control* ON. The *gazeCursor* is a feature of *ViewPoint*, whereas the *mousePointer* is part of the operating system.

A mouseClick event can be issued at to the location of the *gazeCursor*, see: *cursor_DwellClick* and *cursor_BlinkClick*.

`VPX_SendCommand("gazeCursor_Used Toggle");`

19.22.2 GazeCursor Transparency

GUI: *-none-*

CLI : *gazeCursor_transparency intValue*

intValue: integer in range 0 to 255

Default: 128

Controls the transparency of the *gazeCursor*. The larger the value the less transparency.

See also: *gazePoint_Transparency* (that takes a normalized argument)

`VPX_SendCommand("gazeCursor_Transparency 150");`

19.22.3 Turn Cursor Control On / Off

GUI: *Interface > CursorControl > Eye Moves Mouse* ^E

CLI : *cursor_Control BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Off

When ON, the *mousePointer* will be positioned where the subject is looking. This is independent of whether or not the *gazeCursor* is used.

To see a transparent disk where the subject is looking, set: *gazeCursor_Used YES*

`VPX_SendCommand("Cursor_Control On");`



Arrington Research

19.22.4 Use Fixation to Issue Button Click

GUI: *Interface > CursorControl > Fixation Clicks Buttons* *Alt + Shift + E*

CLI : *cursor_DwellClick BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Off

If ON, a *mouseClick* event will be issued whenever the *dwellTime* exceeds the value set for *cursor_DwellSeconds*.

`VPX_SendCommand("cursor_DwellClick On");`

19.22.5 Specify Fixation Time to Issue Button Click

GUI: *Controls window, Criteria tab, MouseClick DwellTime slider*

CLI : *cursor_DwellSeconds FloatValue*

FloatValue: From 0.00 to 9.00.

Sets the amount of time until mouse click is issued due to fixation.

Default 3.5 seconds

Specifies the *dwellTime* in seconds before a *mouseClick* event is issued, if *cursor_DwellClick* ON

`VPX_SendCommand("cursor_DwellSeconds 2.5");`

19.22.6 Use Blinks to Issue Button Click

GUI: *Interface > Cursor Control > Blinks Click Buttons (toggle)*

CLI : *cursor_BlinkClick BoolValue*

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: Off

When this is enabled, blinks will evoke a *mouseClick* event at the location of the *gazeCursor*.

`VPX_SendCommand("cursor_BlinkClick On");`

19.23 Ethernet

19.23.1 Ethernet Server

GUI:	<i>Interface > Ethernet > Server (toggle checkmark)</i>
CLI :	<code>ethernet_server BoolValue</code> <i>BoolValue:</i> Yes, No, True, False, On, Off, 1, 0, Toggle
Default:	On
Normally the server is always running and should not need to be turned off, however if this seems necessary, it is accomplished with this command.	
<code>VPX_SendCommand("ethernet_server Off");</code>	

19.23.2 Ethernet Port Number

GUI:	<i>-none-</i>
CLI :	<code>ethernet_setPortNumber unsignedInteger [-noConfirm]</code>
Default:	5000
The current default Ethernet port is 5000 and should not need to be changed unless a conflict is detected. However if it is changed, the server will be stopped and all client connections will be lost, then the server will be restarted; any clients will need to be attached again manually. <i>ViewPoint</i> will ask you to confirm this unless you include the option: <i>-noConfirm</i>	
<code>VPX_SendCommand("ethernet_setPortNumber 5100");</code>	

19.23.3 Ethernet IP Address

GUI:	<i>-none-</i>
CLI :	<code>ethernet_setIPAddress string [-noConfirm]</code>
Default:	Obtained from the network interface card (NIC)
Rarely does the server's IP Address need to be changed, but there may be occasions when multiple NIC cards are installed or there is an IP address conflict with other network connections. <i>ViewPoint</i> automatically tries to connect with the first NIC listed on the system. During startup, if more than one NIC is discovered, a notice will be reported (as of version 2.9.3.128). If the <i>ViewPoint</i> IP Address is changed, the server will be stopped and all client connections will be lost, then the server will be restarted; any clients will need to be attached again manually. <i>ViewPoint</i> will ask you to confirm this unless you include the option: <i>-noConfirm</i>	
There is no <i>name resolution</i> , so you need to use the IP address not the computer name.	
<code>VPX_SendCommand("ethernet_setIPAddress '192.168.1.33' ");</code>	

19.23.4 Ethernet List IP Addresses

GUI:	<i>-none-</i>
CLI :	ethernet_listIPAddresses
Default:	
Prints information in the <i>History</i> window about each <i>Network Interface Controller</i> (NIC) installed on the computer. These can also be called a: network interface card, network adapter, LAN adapter or physical network interface. You may need to change the Ethernet address that <i>ViewPoint</i> is using to that of another NIC, if more than one is installed on the computer.	
<code>VPX_SendCommand("ethernet_listIPAddresses");</code>	

19.23.5 Ethernet Ping Clients

GUI:	<i>Interface > Ping Clients</i>
CLI :	ethernet_PingClients
Default:	
Instructs <i>ViewPoint</i> to send a Ping message to all the attached clients. Each attached client should respond with a Pong message. When <i>ViewPoint</i> receives a Pong response, it calculates the total round-trip time, and prints this time in the <i>History</i> window. This message includes a unique identifier number for each client connection.	
<code>VPX_SendCommand("ethernet_PingClients");</code>	

19.24 Binocular Commands

19.24.1 Turn Binocular Mode On / Off

GUI:	<i>Binocular > Binocular mode (toggle checkmark)</i>
CLI :	binocular_Mode BoolValue <i>BoolValue:</i> Yes, No, True, False, On, Off, 1, 0, Toggle
Default:	Off
Turns binocular operation mode on or off. When On, the data file will automatically include the data from Eye_B and this data will be available in real-time via the SDK.	
<code>VPX_SendCommand("Binocular_Mode On");</code>	



Arrington Research

19.24.2 Specifies Binocular Averaging

GUI:	<i>Binocular ></i> <i>Show both eye positions</i> <i>Show averaged Y-gaze positions</i> <i>Show average of eye positions</i> <i>Show average with parallax correction</i>
CLI :	<i>binocular_Averaging averageOption</i> <i>averageOption</i> : Off, only_Y, both_XY, ParallaxCorrection
Specifies whether to use binocular averaging and which type.	
<i>Off</i> – No averaging is performed.	
<i>Only_Y</i> : Averages only the y-values, resulting in the same y-value for each eye, while leaving the x-values different. This leaves vergence information while greatly reducing or eliminating errors due to rotation of the head or rotation of the eyeFrames about the bridge of the nose.	
<i>Both_XY</i> : This yields a single point that is the average of the two eye positions.	
<i>ParallaxCorrection</i> : Same as Both_XY, but also applies parallax correction based on vergence.	
<code>VPX_SendCommand("binocular_Averaging both_XY");</code>	

19.24.3 Specifies which Eye to Calibrate

GUI:	<i>EyeSpace window, Pull Down List > { Eye A Only, Eye B Only, Both Eyes }</i>
CLI :	<i>calibration_eyeTarget eyeTraget</i> <i>eyeTraget</i> : EyeA, EyeB, Both
Default:	<i>Both</i>
Specifies whether to calibrate both eyes at the same time or to do one or the other eye separately. This is particularly useful if you already have a good calibration for one eye and you do not want to mess up that calibration while trying to fix the calibration of the other eye.	
<code>VPX_SendCommand("calibration_eyeTarget Both");</code>	

19.25 System Files & Applications Related

19.25.1 Launch Application with Command Line Options

GUI:	<i>-none-</i>
CLI :	<pre>launchApp applicationName argumentsToApp applicationName: The name of the application to launch argumentsToApp: command line arguments may be flags, input/output file names, etc.</pre>
SDK:	<code>VPX_LaunchApp(applicationName, argsToApp);</code>

A general purpose command to launch an application and to provide it with optional arguments.

Note: Must use full path for file names if they are not in the default location for the application.

Special: This command will enable playing stimulus movies.

WARNING: The CLI strings are parsed by the *ViewPoint* application, therefore you cannot use this CLI string to launch the *ViewPoint* application itself, instead use:

`VPX_LaunchApp("ViewPoint.exe", "");`

See also: [systemOpen](#), [quitViewPoint](#), [VPX_LaunchApp](#)

- Example 1

```
VPX_LaunchApp("ViewPoint.exe", ""); // Layered app launched ViewPoint
int running = VPX_GetStatus( VPX_STATUS_ViewPointIsRunning );
// Wait until true, so parser is running.
```

- Example 2

```
VPX_SendCommand ("launchApp ViewPoint.exe"); // WRONG
// PARSE ERROR because ViewPoint (and so the ViewPoint parser) is NOT RUNNING
```

- Example 3

```
fKey_cmd 9 { launchApp C:\Windows\Explorer.exe Data } // Opens the Data Folder
// DANGER: If Explorer.exe is in any other folder
// or subfolder then it is a virus, spyware, trojan or worm!
```

- Example 4

```
launchApp C:\Windows\Explorer . // The dot means the current folder, here ~/ViewPoint/
```



Arrington Research

19.25.2 SystemOpen

GUI: *-Various File menu and other commands-*

CLI : **systemOpen string**

string = fileName, folderName, HTTP-path

General purpose instruction to open a system file, folder, or an HTTP address.

The string must be inside quotes.

WARNING: Currently when using a default folder, such as `./ViewPoint/Data/` you must follow it with a backward-slash before specifying a file name.

See also: [launchApp](#), [VPX_LaunchApp](#)

```
VPX_SendCommand ("systemOpen 'C:\\\\ARI\\\\Misc\\\\myFile.txt' );
```

```
systemOpen "http://www.arringtonresearch.com/torsion.html"
```

```
systemOpen "Data\\123.txt" // The ViewPoint/Data folder
```

19.25.3 Quit ViewPoint

GUI: *File > Quit*

CLI : **quitViewPoint**

Terminates the *ViewPoint EyeTracker* application.

See also: [confirmQuit](#)

```
VPX_SendCommand ("quitViewPoint" );
```

19.25.4 Confirm Quit

GUI: *-none-*

CLI : **confirmQuit BoolValue**

BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: On

Specifies whether *ViewPoint* should display a dialog box that asks the user to confirm a quit request, before terminating the program. It is suggested that this specification be placed in the *Settings* file: `~/ViewPoint/Startup/StartUp.txt`

See also: [quitViewPoint](#)

Added: 2.8.4

```
VPX_SendCommand ("confirmQuit No" );
```

19.25.5 Specify Default Folder Paths

GUI: *-none-*

CLI : *setPath folderID pathString*
folderID: VIEWPOINT: IMAGES: DATA: SETTINGS: SOUNDS:
DOCUMENTATION: CALIBRATION:
pathString : full file path string, or the special term : **DEFAULT_PATH**

Specifies the current default folder path for important *ViewPoint* directories.

The special keyTerm **DEFAULT_PATH** may be used to reset the path to the original *ViewPoint* default path string. Usually this is a folder directly under the main *ViewPoint* folder.

Note: The colon is part of the *folderID* and must be included, e.g., **IMAGES:**

Note: The *pathString* must be in quotes.

Note: paths may be set across machines, for example via mapped network drives, however saving data across such a path may cause delays or data loss.

Accepts both forward slashes and backward slashes, and converts them to forwards

Appends a final slash at the end of the string, if it was not included by the user.

Tests that the path exists and reports an error if it does not.

See also: [VPX_GetViewPointHomeFolder\(pathString \);](#)

```
setPath IMAGES: DEFAULT_PATH
setPath IMAGES: "C:/ARI/Global/Images/"
```

19.26 FKey

19.26.1 Associate CLI's with FKeys

GUI:

-none-

CLI :

Fkey_cmd FKeyNumber { commandString }

FKeyNumber : integer in { 1 to 12 }

commandString : any valid command

Assigns CLI commands to FKeys. It is a very useful to customize the Fkeys for your needs.

These FKey associations can be viewed in the *Info panel* from menu: *Help > Info > ShortCuts tab*

Everything inside the *Deferred Command Block* (i.e. evaluated later) must be in curly braces (see [19.1](#) and [18.7.2](#)). More than one command, separated by semicolons may be inside the curly braces.

Note: Do not put quotes around commands; this method is obsolete and may no longer work properly.

Restore defaults with: **fkey_default**

```
VPX_SendCommand ( "FKey_cmd 12 { dataFile_NewUnique } " ) ;
VPX_SendCommand ( "FKey_cmd 11 { stimulus_playSoundFile 'No Way .wav' } " );
```

19.27 TTL

19.27.1 Associate CLI's with TTL Voltage Changes

GUI: *-none-*

CLI : **ttl_cmd** *signedChannel* { *commandString* }
signedChannel : +/- just before an integer in { 0 to 7, and 99 }
commandString : any valid command

Allows CLI commands to be associated with incoming Digital-Input (TTL) voltage changes, high or low.

These associations can be viewed in the *Info panel* from menu: *Help > Info > TTL Cmds tab*.

Everything inside the *Deferred Command Block*, i.e., inside the curly braces, is saved and evaluated later see [19.1](#) and [18.7.2](#). More than one command, separated by semicolons may be inside the curly braces.

Channel 99 is associated with an EdgeTrigger from the Digital-Input.

Note: Do not put quotes around the TTL command string; this method is obsolete and may no longer work properly.

Restore defaults with: **ttl_default**

Requires the TTL option to send TTL voltages, but can be tested using: **ttl_simulate**

```
VPX_SendCommand ("ttl_cmd +0 { dataFile_Pause On } ");
VPX_SendCommand( "ttl_cmd -0 { dataFile_Pause Off } " );
VPX_SendCommand( "ttl_cmd +1 { dataFile_NewUnique } " );
VPX_SendCommand( "ttl_cmd -1 { say 'TTL ch#1 LO' } " );
VPX_SendCommand( "ttl_cmd +2 { dataFile_InsertMarker 2 }" );
VPX_SendCommand( "ttl_cmd 99 { say 'EdgeTrigger'; stateJump 1 }" );
```

19.27.2 Set TTL Output Voltages

GUI: *-none-*

CLI : **ttl_out** *signedChannel*
signedChannel : +/- just before an integer in { 0 to 7 }

Provides a mechanism to easily set the TTL output voltages.

There must be no space between the sign and the number.

Requires the TTL option.

Be sure to de-select *ViewPoint~Voltage* menu item: *Options > TTL > TTL output of ROI code*

```
VPX_SendCommand( "ttl_out -0" ); // set TTL channel 0 LO
VPX_SendCommand( "ttl_out +7" ); // set TTL channel 7 HI
```

19.27.3 Simulate Change in TTL Input

GUI: *-none-*

CLI : **ttl_simulate signedChannel**
signedChannel : +/- just before an integer in { 0 to 7 }

Simulates a TTL signal coming from the TTL hardware option, regardless of whether or not this hardware is installed. This is very useful for development and debugging.

Does not require the TTL option.

```
VPX_SendCommand( "ttl_simulate +0" ); // simulate TTL channel 0 HI event
```

19.27.4 Print TTL Values in the *History* Window

GUI: *-none-*

CLI : **ttl_values**

The current *TTL-In* and the *TTL-Out* values are reported in the *History* window.

```
VPX_SendCommand( "ttl_values" );
//
// The following is subject to change!
int inValues = VPX_GetStatus( VPX_STATUS_TTL_InValues ); // subject to change
int outValues = VPX_GetStatus( VPX_STATUS_TTL_OutValues ); // subject to change
// The SDK interface returns an integer with the high channels bit coded
// as ones and the low channels bit coded as zeroes.
// May not be available in the future through the VPX_GetStatus function.
```

19.27.5 Set TTL Output to Indicate Data Quality Codes

GUI: *-none-*

CLI : **ttl_out_quality channel levelString**
 channel : integer in { 0 to 7 }
 levelString : OFF, or any of VPX_QUALITY_*
 VPX_QUALITY_PupilScanFailed
 VPX_QUALITY_PupilFitFailed
 VPX_QUALITY_PupilCriteriaFailed
 VPX_QUALITY_PupilFallback
 VPX_QUALITY_PupilOnlyIsGood
 VPX_QUALITY_GlintIsGood

A TTL output channel can be specified to indicate when the data quality value is **greater than or equal to** (\geq) the quality criterion level.

The level strings are identical to the VPX_QUALITY_* constants defined in the [VPX.h](#) file. The best quality is level == [VPX_QUALITY_GlintIsGood](#), poorer quality raises the quality level. Setting the quality criterion to [VPX_QUALITY_PupilScanFailed](#) will cause the TTL channel to always be high, because the data quality value is always greater than or equal to this.

Be sure to de-select *ViewPoint~Voltage* menu item: [Options > TTL > TTL output of ROI code](#)

See also: [VPX_GetDataQuality](#), [verbose +ttl_out](#)

```
VPX_SendCommand( "ttl_out_quality 0 VPX_QUALITY_PupilFallback" );
```

19.28 Misc.

19.28.1 Specify Verbose Information to Send to *History* Window

GUI: *-none-*

CLI : *verbose +/-activityType*

activityType : see list here below

Allows fine control over the type of verbose information sent to the *History* window.

The reports are turned on or off by preceding the keyTerm with a plus (+) or minus (-), respectively.

There can be no space between the sign and the keyTerm.

This is used for debugging and the details of what is reported may change without notice.

The following arguments can be used to turn reporting on or off.

+setting

+parsing

+ttl_out

+ttl_cmd

+frameGrabber

+videoTiming

+server

+insertMark

+insertString

+insertUserTag

+calibration

+nameList // this includes things like the pictureList actions

```
VPX_SendCommand( "verbose -calibration +settings" );
```

19.28.2 Status Dump

GUI: *-none-*

CLI : *status_dump*

Reports the values of various *Status* variable in the *History* window. The variables are in general the same as those available via the SDK function: [VPX_GetStatus](#).

See [VPX_GetStatus](#) for list of [VPX_STATUS_*](#) variables

```
VPX_SendCommand( "status_dump" ); // Dumps values in ViewPoint History window.
```

```
bool calibrating = VPX_GetStatus(VPX_STATUS_CalibrationInProgress);
```



Arrington Research

19.28.3 Update Eye Data on Request

GUI: *Alt-Shift-U*

CLI : *updateData*

Some programs want as much CPU time as they can get, so they would like to have *ViewPoint* video image processing turned off until fresh data is needed. This command updates the eye data based on the most recent video image in memory (on 60 Hz systems this memory is constantly being updated via direct memory access (DMA) by the video capture driver).

Note: Using this assumes that all video is frozen (*videoFreezeSync ON*); sending an *updateData* command while NOT frozen may cause a glitch in the data timing.

```
VPX_SendCommand ( "updateData" );
```

19.28.4 Set Status Window Update Rate for FPS Field

GUI: *-none-*

CLI : *fpsUpdate nth_Interrupt*
nth_Interrupt : integer

Controls for rate of update of the FPS (Frames Per Second) value in the *PenPlot* window. The argument may be any positive number. An argument of 1 would cause the FPS calculation to be updated every video interrupt (frame or field), 2 would be every 2nd, etc.

High speed digital cameras may apply their own data thinning.

```
VPX_SendCommand ( "fpsUpdate 3" );
```

19.28.5 SDK Debug Mode

GUI: *-none-*

CLI : *debugSDK BoolValue*
BoolValue: Yes, No, True, False, On, Off, 1, 0, Toggle

Default: **Off**

This command will add debugging capability for the DLL based SDK.

```
VPX_SendCommand ("debugSDK YES"); // The DLL attached directly to ViewPoint  
VPX_DebugSDK(1); // The DLL that the layered application is talking to.
```



19.28.6 Priority

GUI: *-none-*

CLI : **setPriority priorityLevel**
 priorityLevel: Normal, Above, High, RealTime

Default: **Normal**, but some threads may have higher priority.

See *Microsoft* documentation for: [SetPriorityClass](#)

Added in version 2.9.3.116

`VPX_SendCommand ("setPriority High");`

19.28.7 Specify *ViewPoint* Generated Events

GUI: *-none-*

CLI : **vpx_event +option -option**
 option: videoSynch

This provides a mechanism to control (thin) the number of events that *ViewPoint* sends out. The command **vpx_event** followed by one or more options that are immediately preceded (no spaces) by a + or - character.

Note: unnecessary messages will unnecessarily consume system resources.

See: [VPX_VIDEO_SyncSignal](#)

`VPX_SendCommand ("vpx_event +videoSynch");`

19.29 Parser Instructions

19.29.1 Comment

GUI: *-none-*

CLI : **//**

Use double forward slashes **//** to tell the CLI parser to ignore everything to the right of these slashes. They can be used anywhere on a command line.

Deprecated old CLI command: **COMMENT**

// Ignore all of this comment line

Say "Hi there" // Then ignore this other stuff



Arrington Research

19.29.2 End of Settings File Command

GUI: *-none-*

CLI : **END**

The command **END** should be placed by itself on the final line of a *Settings* file. This provides a way to verify that all the command lines in the file were read. *ViewPoint* reports "Settings read successfully" when this command is reached.

END

Chapter 20. Software Developers Kit (SDK) & API

20.1 General

A third party application (for example, your application) may interact with the *ViewPoint EyeTracker*® by compiling with the `VPX_InterApp.lib` file, a library file. At run time your program will dynamically link to the `VPX_InterApp.DLL`, a Dynamic-Link Library file. All of the *ViewPoint* inter-application communication constants, data types and functions begin with the prefix `VPX_` and are specified in the `VPX.h` file.

Your application may call the `VPX_` routines `VPX_LaunchApp("ViewPoint.exe", " ")` and `VPX_SendCommand("quitViewPoint")` to control when the *ViewPoint EyeTracker*®, application is running. If the remote application will provide its own control settings to *ViewPoint*, then it may be desirable to launch only the *EyeCamera* window, by calling:

```
VPX_LaunchApp("ViewPoint.exe", " -hideMain -freeEyeCamera ") // Case Sensitive Flags
```

The command line argument `-minimized` allows access to the main *ViewPoint* program window via the minimized icon, `-hideMain` makes the main window of *ViewPoint* completely inaccessible.

The flag `-freeEyeCamera` launches *ViewPoint* with the eye camera window as a free floating window.

These flags are case sensitive!

Note: currently, if launched minimized, the splash window is not presented.

Your application may access *ViewPoint* data at any time. It should be noted that employing a tight polling loop is a poor programming practice, because it unnecessarily consumes computer CPU time. If only occasional updates are required, a timer would probably be the preferred method (see *Microsoft Windows SetTimer* function). If your application needs to respond immediately every time the data values are updated, status is changed, etc., then it should register a *CallbackFunction*.

20.2 RealTime Callback Functions

Layered applications can respond to *ViewPoint* data and status changes quickly and easily (i.e., be event driven) by registering a *CallbackFunction*. The function signature (i.e., the number, type and order of the arguments) must be exactly the same as specified, however the users can put whatever they need in the body. Here is an example:

```
int theCallbackFunction( int msg, int subMsg, int param1, int param2, void* userPtr );
int theCallbackFunction( int msg, int subMsg, int param1, int param2, void* userPtr )
{
if ( ( VPX_DAT_FRESH == msg )&& ( EYE_A == subMsg ) ) {
    VPX_RealPoint gp; // a structure with two floats for (x,y) values
    VPX_GetGazePoint2( EYE_A, &gp ); // pass variable by reference
    printf("GazePoint = ( %g, %g ) \n", gp.x, gp.y );
}
VPX_InsertCallback( &theCallbackFunction, this );
```

The `userPtr` is typically used for the C++ `this` value.

20.3 Registering to Receive WindowMessages (Deprecated)

Registering to receive notifications of fresh data is a multi-step process:

Obtain the unique message identifier used by *ViewPoint* for inter-process communication:

```
const static UINT wm_VPX_message = RegisterWindowMessage(VPX_MESSAGE);
```

The window(s) that wish to receive notification must register to receive it by calling:

```
VPX_InsertMessageRequest( m_hWnd, wm_VPX_message );
```

More than one window in an application may request notifications; however no window should make more than one request.

Your program must listen for the notifications. This may be done in several ways. If you are using MFC message maps, then you should add a mapping, e.g.:

```
ON_REGISTERED_MESSAGE( wm_VPX_message, OnVPX_message );
```

If you are using Win32, then you will want to listen for the `wm_VPX_message` notification just as you would listen for a `WM_PAINT` message.

When a `wm_VPX_message` is received, your message handling routine should determine the type of the notification, as follows:

```
WORD notificationCode = HIWORD (wParam);
```

This notificationCode may then be used in a switch statement that uses case constants defined in the `VPX.h` file. Among others, these include: `VPX_DAT_FRESH`, `VPX_ROI_CHANGE`, and notifications relating to the presentation of Calibration Point Stimuli. Refer to section 21.14 for a complete list.

IMPORTANT: A finite number of message windows may register a `VPX_InsertMessageRequest` at one time (currently ten), after which notification requests will be refused. Your application should always make sure that each requesting window successfully removes its request for notification before the window is destroyed or before the program terminates, by calling: `VPX_RemoveMessageRequest(m_hWnd)`.

The *ViewPoint EyeTracker®* Status window contains the field **DLL sharing**. This shows the number of windows that are currently registered to receive notifications. You may monitor this value during program development to make sure that terminating your application also decrements this value. If you find that all requests are used up, the only way to reset this list is to terminate every application that dynamically links to the `VPX_InterApp.DLL`.

Note 1: The `VPX_Set_x` routines work by directly sending values to the *ViewPoint EyeTracker®*. Any commands sent before *ViewPoint* is launched, will have no effect on the initial startup values of *ViewPoint*.

Note 2: *ViewPoint* does not send out notifications when control values are changed, so unless the remote application explicitly sets the *ViewPoint* values, the remote applications control values (e.g. of sliders) will not accurately reflect the *ViewPoint EyeTracker®* application control values.

We encourage users and third party developers to work with us in developing this interface. We take suggestions and usability reports very seriously.



20.3.1 Example SDK Code (Deprecated method)

Your application must first register with the VPX_InterApp.lib by doing the following for each window procedure (WinProc) that desires notification:

```
if ( uniqueMessageId == 0 ) {
    uniqueMessageId = RegisterWindowMessage( VPX_MESSAGE );
    VPX_InsertMessageRequest( hWnd, uniqueMessageId );
}
```

After registering, the WinProc should listen for notifications:

```
LRESULT CALLBACK WndProc ( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam )
{
    WORD notificationCode = HIWORD(wParam);
    if ( message == uniqueMessageId ) {
        switch ( notificationCode )
        {
            case VPX_DAT_FRESH :
            {
                EyeType eye = LOWORD ( wParam );
                showFreshData( eye graphicsWindow );

            } break;
        }
    }
}
```

20.4 Data Quality Codes

A `VPX_DAT_FRESH` is sent and the SDK application is responsible for obtaining the quality code and making a decision as to what levels of quality are appropriate for various situations. We highly recommend that you use the constants provided, rather than testing the integer value, as these are not guaranteed to remain unchanged. Nevertheless, the hierarchical relationships are expected to remain intact. For example:

```
VPX_GetDataQuality2( eye, &quality );
switch (quality)
{
    case VPX_QUALITY_PupilScanFailed: showGaze(false); break;
    case VPX_QUALITY_PupilFitFailed: showGaze(false); break;
    case VPX_QUALITY_PupilCriteriaFailed: showGaze(false); break;
    case VPX_QUALITY_PupilFallBack: showGaze(true ); break;
    case VPX_QUALITY_PupilOnlyIsGood: showGaze(true ); break;
    case VPX_QUALITY_GlintIsGood: showGaze(true ); break;
}
```

Data records are constantly stored, because the data quality may vary with different sources. For example, data may be good from one eye, and a wink occurred in the other eye, or both eyes are closed, but other data is still good. Data sections now contain individual quality columns, as needed.

20.5 Sending CLI's via the SDK

The *ViewPoint EyeTracker® Software Developers Kit (SDK)* provides a routine that allows sending *Command Line Interface (CLI)* commands directly to the *ViewPoint EyeTracker*. This means that the same



command strings that are loaded from *Settings* files can be issued via the SDK either on the same machine or via Ethernet from another machine. An important benefit is that this allows passing textStrings and fileNames.

Note: *ViewPoint* expects 8-bit chars.

```
int result = VPX_SendCommand( char *cmd );
VPX_SendCommand("dataFile_NewName rabbitPictureData");
VPX_SendCommand("dataFile_InsertString \"Showing Picture of a Rabbit \" ");
VPX_SendCommand("settingsFile_Load RabbitPictureROI.txt");
```

The function returns an integer result value that provides feedback about the success or problems encountered. See [VPX.h](#) for a list of return codes.

Note: All `VPX_Set*` functions are deprecated in favor of the single `VPX_SendCommand` function.

20.6 High Precision Timing

High Precision timing (HPT) with resolution in the order of 0.0000025, i.e., 2.5E-6 or 2.5 microseconds, is available via the SDK function call.

The function returns the difference between the `pdInHoldSeconds` argument and the current time. This function will automatically update the `pdInHoldTime` with the current time for use with the next call, if `iFlags` is `RESET_PRECISE_HOLD_TIME`. When `SINCE_PRECISE_INIT_TIME` is passed as the `pInHoldSeconds`, it returns the time since the initialization of either the DLL of the System, depending upon the second argument.

The four combinations are shown here below:

```
secondsSinceLastSomeEvent =
    VPX_GetPrecisionDeltaTime( &(pv.someEventTime),           RESET_PRECISE_HOLD_TIME );

secondsSinceLaunchingTheApp =
    VPX_GetPrecisionDeltaTime( &(pv.appLaunchTime),           LEAVE_PRECISE_HOLD_TIME );

secondsSinceStartingTheDLL =
    VPX_GetPrecisionDeltaTime( SINCE_PRECISE_INIT_TIME,      SINCE_DLL_INIT_TIME );

secondsSinceSystemBooted =
    VPX_GetPrecisionDeltaTime( SINCE_PRECISE_INIT_TIME,      SINCE_SYSTEM_INIT_TIME );
```

20.7 DLL Version Checking

All applications using the *ViewPoint* SDK should include the following check for a possible mismatch between the version of the DLL (loaded at runtime) and the version of the SDK library (prototypes and constants) that was compiled into the application. Note that as of version 2.9.3.115, only the first three values of the version quad are checked, the last value is for small patches that do not effect interapplication communication.

```
BOOL versionMismatch = VPX_VersionMismatch (VPX_SDK_VERSION) ;
double dllVersion = VPX_GetDLLVersion();
if ( VPX_SDK_VERSION != dllVersion ) doSomething();
```

20.8 SDK Trouble Shooting

This section describes some of the common problems users encounter when programming with the SDK and how to solve them.

20.8.1 Different DLLs

The most common problem is when the user makes a copy of the DLL, such that *ViewPoint* is putting data into the DLL that is attached to it, while the layered application is trying to get data from another DLL that is attached to that. Imagine two people agree to meet in *Central Park*, but one is in *Central Park New York*, while the other is in *Central Park Denver*.

It is possible for a layered application to use a separate DLL, either on the same machine or on a different machine, but the separate DLL must establish a link with *ViewPoint* via the Ethernet.

The symptom is that the layered application only gets zero values from the separate DLL.

There are two solutions: (i) make sure that *ViewPoint* and the layered application are using same DLL, or (ii) connect the separate DLL to *ViewPoint* via Ethernet using the *ViewPointServer* and *ViewPointClient* software included.

Select menu item: *Help > Info > SysInfo tab* and look at the *DLL path:* line to view the full path file name of the DLL that is loaded and being used by the *ViewPoint* application.

20.8.2 Different DLL Versions

DLL version changes are reflected in whole number program version changes. Changes in the decimal part of the version number should only reflect bug fixes, patches, and small incremental changes. In general, even with a version mismatch, many of the functions will still work, but the user should verify that the results are accurate.

20.9 SDK Data Access Functions

There are many functions provided to access *ViewPoint* data and the current state of the *ViewPoint* program. The *ViewPoint PenPlot* window uses many of these functions to access the data that is plotted, so this can be used for validation.

Some of these functions have dual roles – the returned values may be different if the *3DViewPoint* or *3DWorkSpace* products are being run; details are described in the documentation for these products.

Currently, the legacy monocular only functions still exists and return the value for EYE_A; but these are deprecated in favor of the newer binocular functions that generally end with the number 2 and require that Eye_A or Eye_B is specified, for example:

```
VPX_GetSomething( &something ); // Always returns EYE_A value.
VPX_GetSomething2( EYE_B, &something );
```

Chapter 21. SDK Functions

21.1 GazeData

21.1.1 GetGazePoint

```
int VPX_GetGazePoint( VPX_RealPoint* gp ); // EYE_A
int VPX_GetGazePoint2( VPX_EyeType eyn, VPX_RealPoint* gp );
int VPX_GetGazePointSmoothed( VPX_RealPoint* gp );
int VPX_GetGazePointSmoothed2( VPX_EyeType eyn, VPX_RealPoint* gp );
int VPX_GetGazePointCorrected2( VPX_EyeType eyn, VPX_RealPoint* gp );
```

Units: Normalized *GazeSpace* coordinates.

Retrieves the calculated Position of Gaze (POG).

Monocular *ViewPoint* uses Eye_A by default.

GazePoint returns the pure POG, GazePointSmoothed returns the smoothed POG, and GazePointCorrected returns the corrected POG that may include smoothing, averaging, parallax correction, nudging, etc...

Returns 1.

*** Data comes from *3DViewPoint* or *3DWorkSpace* when using these products and data will be different.

See also: [VPX_GetPupilPoint](#), [VPX_GetGlintPoint](#)

```
VPX_RealPoint gp ;
VPX_GetGazePointCorrected2( EYE_A, &gp );
printf( "X: %g , Y: %g ", gp.x, gp.y );
```

21.1.2 GetGazeBinocular

```
int VPX_GetGazeBinocular( VPX_RealPoint *gb );
```

Units: Normalized *GazeSpace* coordinates.

This is included to provide a single function that will work well regardless of Monocular/Binocular settings – it provides “one stop shopping”, so layered code doesn’t need to worry about the *ViewPoint* configuration.

Retrieves the corrected Binocular Position of Gaze (POG) in normalized coordinates. The Binocular POG is calculated using the *GazePointCorrected* values from EYE_A and EYE_B. The Binocular POG value and data fresh update are dependent on the binocular mode and averaging setting. Below are the rules.

Monocular or no binocular averaging:

Value same as EYE_A; EYE_A does the update.

Binocular and binocular averaging has 2 gaze points (averaging Y only):

Value same as EYE_A; EYE_A does the update.

Binocular and binocular averaging has 1 gaze point (averaging X/Y with/without parallax):

Value is the averaged EYE_A and EYE_B POG; EYE_A and EYE_B both do the update.

Caution: THERE MAY BE TWICE AS MANY UPDATES IN THIS MODE.

Returns 1.

See also: [VPX_GetGazePointCorrected2](#), [VPX_GetVelocityBinocular](#)

```
VPX_RealPoint gb ;
VPX_GetGazeBinocular( &gb );
printf( " X: %g , Y: %g ", gb.x, gb.y );
```

21.1.3 GetGazeAngle

```
int VPX_GetGazeAngle2( VPX_EyeType eyn, VPX_RealPoint *ga );
int VPX_GetGazeAngleSmoothed2( VPX_EyeType eyn, VPX_RealPoint *ga );
int VPX_GetGazeAngleCorrected2( VPX_EyeType eyn, VPX_RealPoint *ga );
```

Units: Degrees

Retrieves the calculated Angle of Gaze (AOG) in degrees.

GazeAngle returns the pure AOG, GazeAngleSmoothed returns the smoothed AOG, and GazeAngleCorrected returns the corrected AOG that may include smoothing, averaging, parallax correction, nudging, etc...

Note: In 2D products, these angles are from trigonometric calculations based on the values that the user has measured and set in the *Geometry* window, for the *Stimulus* window size (horizontal & vertical) and the viewing distance. This *Geometry* window is NOT for use with *3DWorkSpace* or *3DViewPoint*.

*** *3DWorkSpace* & *3DViewPoint* set these values when then are running.

Returns 1.

See also: [VPX_GetMeasuredScreenSize](#), [VPX_GetMeasuredViewingDistance](#)

```
VPX_RealPoint ga ;
VPX_GetGazeAngleCorrected2( EYE_A, &ga );
printf( " X: %g , Y: %g ", ga.x, ga.y );
```

21.1.4 GetTotalVelocity

```
int VPX_GetTotalVelocity( double* velocity ); // EYE_A *velocity );
int VPX_GetTotalVelocity2( VPX_EyeType eyn, double* velocity );
```

3DViewPoint & 3DWorkSpace units: degrees per second.

ViewPoint units: qualitative derivative of normalized *GazeSpace* coordinates.

Retrieves the *TotalVelocity* of movement in the (x,y) plane. That is, the first derivative of the **(corrected)** POG for either Eye_A, or Eye_B if binocular mode is on.

Unlike the *ComponentVelocities*, the *TotalVelocity* is positive in any direction. You can obtain a total velocity from the *ComponentVelocities* using the Pythagorean theorem.

Monocular *ViewPoint* uses Eye_A by default.

3DWorkSpace & 3DViewPoint products set these values when they play back a 3D data file, but they are set by *ViewPoint* when it is sending data to the 3D products.

Returns 1.

See also: [VPX_GetComponentVelocity](#), [VPX_GetVelocityBinocular](#)
[VPX_GetGazePointCorrected2](#), [VPX_GetFixationSeconds](#)

```
double velocity ;
VPX_GetTotalVelocity2 ( EYE_A, &velocity );
printf( " Velocity: %g ", velocity );
```

21.1.5 GetComponentVelocity

```
int VPXGetComponentVelocity( VPX_RealPoint *velocityComponents );
int VPXGetComponentVelocity2( VPX_EyeType eye, VPX_RealPoint *veloComponents );
```

3DViewPoint & 3DWorkSpace units: degrees per second.

ViewPoint units: qualitative

Retrieves the x- and y-components of the eye movement velocity for either Eye_A, or Eye_B if binocular mode is on. Monocular *ViewPoint* uses Eye_A by default.

Unlike *TotalVelocity* the *ComponentVelocities* can be positive or negative depending on the direction.

3DWorkSpace & 3DViewPoint products set these values when they are running.

Returns 1.

See also: [VPX_GetTotalVelocity](#), [VPX_GetVelocityBinocular](#)

```
VPX_RealPoint cv ;
VPXGetComponentVelocity2( Eye_A, &cv );
printf( " dx/dt: %g , dy/dt: %g ", cv.x, cv.y );
```



21.1.6 GetVelocityBinocular

```
int VPX_GetVelocityBinocular( double *velocity ) ;
```

Retrieves the total *BinocularVelocity* of movement in the (x,y) plane. That is, the first derivative of the (**corrected**) POG from both Eye_A and Eye_B. The *BinocularVelocity* value and data fresh update are dependent on the binocular mode and averaging setting.

Below are the rules.

Monocular or no binocular averaging:

Value same as EYE_A; EYE_A does the update.

Binocular and binocular averaging has 2 gaze points (averaging Y only):

Value same as EYE_A; EYE_A does the update.

Binocular and binocular averaging has 1 gaze point (averaging X/Y with/without parallax):

Value is calculated from either EYE_A or EYE_B POG; EYE_A and EYE_B both do the update.

Returns 0 on success, non-zero otherwise.

See also: [VPX_GetTotalVelocity](#), [VPX_GetComponentVelocity](#)

```
double velocity ;
VPX_GetVelocityBinocular ( &velocity );
printf( " Velocity: %g ", velocity );
```

21.2 3D-Data

21.2.1 GetHeadPositionAngle

```
int VPX_GetHeadPositionAngle ( VPX_PositionAngle *hpa );
```

Subject to Change!

3DWorkSpace & 3DViewPoint products only.

Retrieves the 6DOF (x,y,z,roll,pitch,yaw) data values.

CRITICAL: User MUST call the `VPX_InitializeSDK` macro at application startup to setup the SDK version and `VPX_PositionAngle` size; otherwise this function will return `VPX_ERROR_InvalidStructSize` or `VPX_ERROR_InvalidStructVersion`.

Returns `VPX_NO_ERROR` on success, one of `VPX_SDKFunctionResult` errors otherwise.

Currently returns one of:

`VPX_ERROR_InvalidArgPtr` // hpa is NULL.

`VPX_ERROR_InvalidStructSize` // SDK structure sizeof `VPX_PositionAngle` does not match the DLL.

You must call `VPX_InitializeSDK` and build your application with the same version as the DLL you are loading.

`VPX_ERROR_InvalidSDKVersion` // SDK version does not match the DLL. You must call

`VPX_InitializeSDK` and build your application with the same SDK version as the DLL you are loading.

See also: [VPX_PositionAngle](#)

```
VPX_PositionAngle hpa;
hpa.version = VPX_SDK_VERSION;
hpa.size = sizeof(VPX_PositionAngle);
VPX_GetHeadPositionAngle ( &hpa );
```

21.2.2 GetPanelHit

```
int VPX_GetPanelHit ( VPX_EyeType eyn, int *panelNumber );
```

3DWorkSpace & 3DViewPoint products only.

Retrieves the ID number of the panel intersected by the *EyeRay*, for each eye, or 0 if no panel is hit. Given multiple panel planes, this returns the first (closest) panel that was hit, for each eye. In general, this is the panel at which the subject is looking.

Note: The EyeRays from the two eyes can hit different panels (as with with stereo HMD display panels).

Note: The panel intersected by the *VersionRay* is not currently calculated (Future).

Returns 1.

```
int panelNumber;
VPX_GetPanelHit ( EYE_A, &panelNumber );
printf( " Hit Panel Num: %d ", panelNumber );
```

See also the 3DWorkSpace / 3DViewPoint documentation for Panel Hit rules.

21.2.3 GetVergenceAngle

```
int VPX_GetVergenceAngle ( VPX_RealType *angleDeg );
```

3DWorkSpace & 3DViewPoint products primarily; simplified calculation in Binocular ViewPoint.

Retrieves the vergence angle in degrees.

0.0 indicates there is no vergence angle.

The vergence angle is only valid in binocular mode when there is an intersection between the eye rays.

Binocular ViewPoint.

ViewPoint uses a simplified calculation that uses only the x-values (projection onto the coronal plane).

ViewPoint sets this to zero if Binocular mode is off.

ViewPoint assumes **EyeA is the RightEye** and that the **Geometry > 2D measurements are in millimeters**.

Returns 1.

```
VPX_RealType angleDeg;
VPX_GetVergenceAngle ( &angleDeg );
printf( " Angle: %g ", angleDeg );
```

21.2.4 GetGazePoint3D

```
int VPX_GetGazePoint3D ( VPX_RealPoint3D *gazePointCentimeters );
```

3DWorkSpace & 3DViewPoint products primarily; simplified calculation in **Binocular ViewPoint**.

Retrieves the verged gaze point, in centimeters, in the global coordinate system.

(-1.0, -1.0, -1.0) indicates there is no verged gaze point.

The verged gaze point is only valid in binocular mode when there is an intersection between the eye rays.

Binocular ViewPoint.

ViewPoint uses a simplified calculation that uses only the x-values (projection onto the coronal plane).

ViewPoint sets all values to zero if Binocular mode is off.

ViewPoint sets the z-distance to -99999.0 when the eye rays are parallel.

ViewPoint assumes **EyeA** is the **RightEye** and that the **Geometry > 2D measurements** are in millimeters.

Remember the **ViewPoint** Geometry window settings are in millimeters, but the **VPX_GetGazePoint3D** routine sets **VPX_RealPoint3D** values in centimeters, so a 100 mm distance screen will be reported as 10 cm.

Returns 1.

```
VPX_RealPoint3D gpCM; // in centimeters (CM)
VPX_GetGazePoint3D ( &gpCM );
printf( " X: %g , Y: %g, Z: %g ", gpCM.x, gpCM.y, gpCM.z );
```

21.2.5 GetVersionAngle

```
int VPX_GetVersionAngle ( VPX_RealPoint *angleDegrees );
```

3DWorkSpace & 3DViewPoint products only.

Retrieves the azimuth (x) and elevation (y) component angles, in degrees, of the version ray projected onto the horizontal and sagittal planes of the head, respectively, that is, the component angles between the nose-ray and the version-ray.

Negative azimuth is left of the sagittal plane, negative elevation is below the horizontal plane.

The version angles are only valid in binocular mode when there is an intersection between the eye rays.
Returns 1.

```
VPX_RealPoint angleDeg;
VPX_GetVersionAngle ( &angleDeg );
printf( " Azimuth: %g , Elevation: %g ", angleDeg.x, angleDeg.y );
```

21.2.6 GetVersionComponentVelocity

```
int VPX_GetVersionComponentVelocity ( VPX_RealPoint *velDegPerSec );
```

3DWorkSpace & 3DViewPoint products only.

Retrieves the azimuth (x) and elevation (y) components of the version ray movement velocity in degrees per second.

Positive is rightward and upward, negative is leftward and downward.

The version component velocity is only valid in binocular mode when there is an intersection between the eye rays.

Returns 1.

```
VPX_RealPoint velDegPerSec;
VPX_GetVersionComponentVelocity ( &velDegPerSec );
printf(" AzimuthVel: %g , ElevationVel: %g ", velDegPerSec.x, velDegPerSec.y );
```

21.2.7 GetVersionTotalVelocity

```
int VPX_GetVersionTotalVelocity ( double *velDegPerSec );
```

3DWorkSpace & 3DViewPoint products only.

Retrieves the Total Velocity of movement of the version ray in degrees per second. That is, the first derivative of the 3D gaze ray. The value is always zero or positive (non-negative).

The version total velocity is only valid in binocular mode when there is an intersection between the eye rays.

Returns 1.

```
double velDegPerSec;
VPX_GetVersionTotalVelocity ( &velDegPerSec );
printf( " Velocity: %g ", velDegPerSec );
```

21.3 Eye Events

21.3.1 GetFixationSeconds

```
int VPX_GetFixationSeconds ( double* seconds );
int VPX_GetFixationSeconds2 ( VPX_EyeType eyn, double* seconds );
```

Retrieves the number of seconds that the *TotalVelocity* has been below the *VelocityCriterion* and the gaze drift has been below the *DriftCriterion*.

A zero (0.0) value indicates a saccade is occurring.

This function replaces the less precise function: VPX_GetFixationDuration(DWORD);

Returns 1.

See also: [VPX_GetTotalVelocity](#), [VPX_GetDrift2](#)

```
double seconds, milliseconds, microseconds ;
VPX_GetFixationSeconds2 ( EYE_A, &seconds );
milliseconds = 1000.0 * seconds ;
microseconds = 1000.0 * milliseconds ;
```

21.3.2 GetDrift

```
int VPX_GetDrift2( VPX_EyeType eyn, double *drift );
```

3DViewPoint & *3DWorkSpace* units: Degrees.

ViewPoint units: Normalized *GazeSpace* coordinates

Retrieves the total 2D drift vector in normalized screen units, normalized with respect to the horizontal dimension.

3DWorkSpace & *3DViewPoint* products set these values when they are running.

Returns 1.

See also: [VPX_GetTotalVelocity](#); [VPX_GetFixationSeconds2](#)

```
double drift ;
VPX_GetDrift2 ( EYE_A, &drift );
printf( " Drift: %g ", drift );
```

21.3.3 GetBlinkEvent

```
VPX_EyeEventType VPX_GetBlinkEvent2( VPX_EyeType eyn )
```

Retrieves the Blink Event .

Returns only one of the following events: VPX_EVENT_NoBlink, VPX_EVENT_Blink_Start, VPX_EVENT_Blink_Continue, or VPX_EVENT_Blink_Stop.

See also: [VPX_EyeEventType](#)

```
VPX_EyeEventType be = VPX_GetBlinkEvent2( EYE_A );
switch ( be )
{
    case VPX_EVENT_NoBlink: break;
    case VPX_EVENT_Blink_Start: break;
    case VPX_EVENT_Blink_Continue: break;
    case VPX_EVENT_Blink_Stop: break;
}
```

21.3.4 GetEyeMovementEvent

```
VPX_EyeEventType VPX_GetEyeMovementEvent2( VPX_EyeType eyn )
```

Retrieves the Eye Movement Event, see example below:

Note that this never returns :

```
VPX_EVENT_Fixation_Stop, VPX_EVENT_Saccade_Stop, VPX_EVENT_Drift_Stop.
```

3DWorkSpace & 3DViewPoint products set these values when then are running.

See also: [VPX_EyeEventType](#)

```
VPX_EyeEventType me = VPX_GetEyeMovementEvent2 ( EYE_A );
switch ( me )
{
    case VPX_EVENT_Fixation_Start: break;
    case VPX_EVENT_Fixation_Continue: break;
    case VPX_EVENT_Saccade_Start: break;
    case VPX_EVENT_Saccade_Continue: break;
    case VPX_EVENT_Drift_Start: break;
    case VPX_EVENT_Drift_Continue: break;
}
```

21.4 ROI

21.4.1 GetROI_RealRect

```
int VPX_GetROI_RealRect ( int n, VPX_RealRect *rr );
```

Retrieves the normalized floating point coordinates for the specified Region of Interest (ROI).

Parameter **n** is the ROI index from { [MIN_ROI_INDEX](#) to [MAX_ROI_INDEX](#) }.

Returns [-998](#) for an invalid ROI index, [1](#) otherwise.

```
// For a known hwnd.
VPX_RealRect rr;
RECT cr;
INT w, h, ix;
GetClientRect( hwnd, &cr ); w = cr.right; h=cr.bottom;
for ( ix = 0; ix < MAX_ROI_BOXES; ix++ )
{
    VPX_GetROI_RealRect( ix, &rr );
    printf("ROI %d = (%d,%d)(%d,%d)",
        (int) (w*rr.left), (int) (h*rr.top), (int) (w*rr.right), (int) (h*rr.bottom) );
}
```

21.4.2 ROI_GetHitListLength

```
int VPX_ROI_GetHitListLength ( VPX_EyeType eyn );
```

Returns the count of ROI that the POG is simultaneously within for the specified eye.

Since ROI can be overlapped or nested, more than one ROI can be active at the same time. This value is NOT cumulative over time.

Returns 0 when the POG is not in any ROI, the count of active ROI otherwise. For example, it returns 3, then three ROI were simultaneously hit. Use [VPX_ROI_GetHitListItem](#) to find out which ROI were hit.

*** Data comes from *3DViewPoint* or *3DWorkSpace* when using these products and data will be different.

See sample code under: [VPX_ROI_GetHitListItem](#)

```
int numberOfRegionsHit = VPX_ROI_GetHitListLength ( EYE_A );
```

21.4.3 ROI_GetHitListItem

```
int VPX_ROI_GetHitListItem ( VPX_EyeType eyn, int NthHit );
```

NthHit is a positive integer between 0 and (MAX_ROI_INDEX – 1)

Returns the ROI index number of the NthHit ROI that the POG is in for the specified eye.

Since ROI can be overlapped, more than one ROI can be active at the same time.

The NthHit argument starts at 0. This function may be called repeatedly in a while-loop until ROI_NOT_HIT is returned, or it can be called from within a for-loop using the count from

[VPX_ROI_GetHitListLength](#).

Active ROI are when the POG is inside the ROI box, not resting on the box lines. The test is for values inside the ROI box values, not resting on the box lines.

Returns -9998 for an invalid NthHit list index, [ROI_NOT_HIT](#) if NthHit is greater than or equal to the hit count or the ROI index number {MIN_ROI_INDEX to MAX_ROI_INDEX} otherwise.

*** Data comes from *3DViewPoint* or *3DWorkSpace* when using these products and data will be different.

See also:

[VPX_ROI_GetHitListLength](#)

[VPX_ROI_GetEventList](#)

```
int roiNumber, ix = 0;
while( ROI_NOT_HIT != ( roiNumber = VPX_ROI_GetHitListItem( EYE_A, ix++ ) ) )
{
    // Do something
}

// or...

// get the number of ROI simultaneously hit
int count = VPX_ROI_GetHitListLength ( EYE_A );
// if at least one ROI was hit, get them from the list of ROI that were hit
if ( count >= 1 )
{
    // indexes into the list just created to get the roiNumber just hit.
    for( int ix = 0; ix < count; ix++ )
    {
        int roiNumber = VPX_ROI_GetHitListItem ( EYE_A, ix );
    }
}
```

21.4.4 ROI_MakeHitListString

```
int VPX_ROI_MakeHitListString ( EyeType eyn , char *dataString,
    int maxStringLength, BOOL indicateOverflow, char *noHitsString );

int VPX_ROI_MakeHitListString_EDR ( VPX_EyeDataRecord* pEyeDataRecord, char
*dataString, int maxStringLength, BOOL indicateOverflow, char *noHitsString );
```

Makes a string that lists the ROI that were hit. The _EDR function allows users to specify the VPX_EyeDataRecord directly if using the dll buffering since the structure contains the ROI hit list. If (indicateOverflow == true) then a "+" at the end of the string will be used to indicate that there were additional ROI that could not fit in the given string.

Makes a string with at most maxStringLength characters. Specifying a maxStringLength of 2 will effectively limit the reported ROI to the first. For double digit numbers, prints both digits if possible, otherwise prints nothing. Does not leave dangling comma separators.

If there are no ROI hit, then the caller's noHitsString is used.

Returns 0 on an error, the length of the string otherwise.

See also: [VPX_ROI_GetHitListLength](#)

Example String: "2,45,88"

```
char myString [80]= "";
VPX_ROI_MakeHitListString( EYE_A, myString, 80, true, "No ROI were hit" );
```

21.5 EyeSpace

21.5.1 GetPupilSize

```
int VPX_GetPupilSize ( VPX_RealPoint *dims ); // EYE_A
int VPX_GetPupilSize2 ( VPX_EyeType eyn, VPX_RealPoint *dims );
```

Retrieves size of the fit to the pupil in Normalized *EyeSpace* units

As of version 2.8.4.524, the major-axis and the minor-axis, ONLY when using the *Ellipse* pupil segmentation method, are both normalized by the width of the EyeCamera, so the scales of the axes are commensurable.

Prior to this version, the x- and y-size values were normalized with respect to the *EyeSpace* dimensions that have a 4:3 aspect, so the x- and y-values were incommensurate. To obtain the aspect ratio of the pupil, you had to rescale: (aspect = ps.x / (ps.y * 0.75))

Pupil Size calculations are relative to which pupil segmentation method is active.

Ellipse - dims.x is the major-axis (larger value) and dims.y is the minor-axis (smaller value). The major-axis and the minor-axis are both normalized by the width of the EyeCamera window so the scales of the axes are commensurable. This is by far the best mode for pupil size estimation.

Oval Fit - dims.x is the HORIZONTAL dimension and dims.y is the VERTICAL dimension of the rectangular bounding box of the unrotated Oval. The HORIZONTAL and VERTICAL dimensions are normalized independently by the width and height of the EyeCamera window respectively, so the scales of the axes are incommensurable. Note that OvalFit is expected to be deprecated in future versions; it still remains only because the current blink detection algorithm performs best in this mode.

Centroid - dims.x and dims.y are 0, because no pupil size is obtained. Note that this method is susceptible to errors from improper lighting and glints over the pupil.

Returns 1.

See also: [VPX_GetPupilDiameter](#), [VPX_GetPupilAspectRatio](#), [VPX_GetPupilOvalRect2](#)

```
double num, dem, aspectCalculated, aspectRetrieved, diff ;
VPX_RealPoint dims ;
VPX_GetPupilSize( &dims );

// Ellipse pupil segmentation method (commensurable).
num = dims.y; // minor-axis
dem = dims.x; // major-axis

// Oval Fit pupil segmentation method (incommensurable).
dims.y *= 0.75; // Scale to make commensurable.
num = std::min ( dims.x, dims.y ); // min
dem = std::max ( dims.x, dims.y ); // max

aspectCalculated = num / dem;
```

21.5.2 GetPupilAspect

```
int VPX_GetPupilAspectRatio( double *ar ); // EYE_A
int VPX_GetPupilAspectRatio2( VPX_EyeType eyn, double *ar );
```

Retrieves the dimensionless value of pupil circularity. This ratio value is independent of the EyeCamera window shape. A perfectly circular pupil will produce a value of 1.0. The Pupil Size x- and y-values are swapped so that the numerator is always less than denominator guaranteeing an Aspect Ratio always in the range {0.0 to 1.0}.

Returns 1.

See also: [VPX_GetPupilDiameter](#), [VPX_GetPupilSize](#), [VPX_GetPupilOvalRect](#)

```
double pupilAspect;
int result = VPX_GetPupilAspectRatio2( EYE_A, &pupilAspect );
// See more details in the example for VPX\_GetPupilSize
```

21.5.3 GetPupilOvalRect

```
int VPX_GetPupilOvalRect ( VPX_RealRect *ovalRect ); // EYE_A
int VPX_GetPupilOvalRect2 ( VPX_EyeType eyn, VPX_RealRect *ovalRect );
```

Retrieves the rectangle, in normalized units, with respect to the EyeCamera window width and height that specifies the OvalFit to the pupil.

Separate rectangles are available for Eye_A or Eye_B if binocular mode is on.

Note: Only valid for the *Oval Fit* pupil segmentation method, set to (0,0,0,0) for the *Ellipse* fitting requires more values to be specified, see: [VPX_GetPupilAngle2](#) method.

Returns 1.

See also: [VPX_GetPupilDiameter](#), [VPX_GetPupilSize](#),
[VPX_GetPupilAspectRatio](#), [VPX_RealRect2WindowRECT](#)

```
// Remote painting of the pupil size and location for a known hWnd.
HDC hDC = GetDC( hWnd );
RECT cr, pr_pixels ;
VPX_RealRect pr_norm ;
VPX_GetPupilOvalRect ( &pr_norm ); // Bounding rectangle
GetClientRect ( hWnd, &cr );
// left & top are zero. The right & bottom are width & height of the window.
VPX_RealRect2WindowRECT ( pr_norm, cr, &pr_pixels );
Rectangle ( hDC, pr_pixels.left, pr_pixels.top, pr_pixels.right, pr_pixels.bottom );
ReleaseDC ( hWnd, hDC );
```

21.5.4 GetPupilAngle

```
int VPX_GetPupilAngle2 ( VPX_EyeType eyn, double *pa );
```

Retrieves the angle or rotation, in degrees, of the rotated ellipse that is fit to the pupil.

Ellipse method: 0-degrees is a vertical ellipse, major-axis from 12- to 6-o'clock; 90-degrees is horizontal, major-axis from 3- to 9-o'clock.

OvalFit method: the angle is always either 0-degrees or 90-degrees since the pupil is never rotated.

Centroid method: the angle is always 0, not a valid value.

To draw the rotated ellipse, (a) get the center via `VPX_GetPupilPoint`, (b) get the major and minor axis lengths via `VPX_GetPupilSize`, (c) get the angle of rotation of the ellipse via `VPX_GetPupilAngle2`.

3DWorkSpace & 3DViewPoint products set these values when they play back a 3D data file, but they are set by *ViewPoint* when it is sending data to the 3D products.

Returns 1.

See also: `VPX_GetPupilPoint`, `VPX_GetPupilSize`

```
double angle ;
VPX_GetPupilAngle2 ( EYE_A, &angle );
printf( " Angle: %g ", angle );
```

21.5.5 GetPupilDiameter

```
int VPX_GetPupilDiameter2 ( VPX_EyeType eyn, double* pdmm );
```

Retrieves the pupil diameter, in millimeters.

The Pupil Diameter is always the larger of the pupil size (x, y) dimensions regardless of the pupil segmentation method selected. The Pupil Scale factor converts normalized units to millimeters. The result will not be correct unless this calibration is performed correctly in advance. The *Pupil Scale factor* is set for each eye in the *Geometry window > PupilScale tab* (see section [9.3](#)).

Returns 1.

See also: `VPX_GetPupilSize`, `VPX_GetPupilAspectRatio`, `EyeA:pupilScaleFactor`

```
double diameter ;
VPX_GetPupilDiameter2 ( EYE_A, &diameter );
printf( " Diameter: %g mm ", diameter );
```

21.5.6 GetPupilPoint

```
int VPX_GetPupilPoint ( VPX_RealPoint *pp ); // EYE_A
int VPX_GetPupilPoint2 ( VPX_EyeType eyn, VPX_RealPoint *pp );
```

Retrieves the **raw** normalized (x,y) location of the center of the pupil (*center of the oval / ellipse fit to the pupil*) in the *EyeSpace*.

Note that this may not be equal to the value obtained via [VPX_GetPupilCentroid2](#).

3DWorkSpace & 3DViewPoint products set these values when they play back a 3D data file, but they are set by *ViewPoint* when it is sending data to the 3D products.

Returns 1.

```
VPX_RealPoint pp;
VPX_GetPupilPoint2 ( EYE_A, &pp );
printf( " X: %g , Y: %g ", pp.x, pp.y );
```

21.5.7 GetPupilCentroid

```
VPX_GetPupilCentroid2 ( VPX_EyeType eyn, VPX_RealPoint *pc );
```

Retrieves the **raw** normalized centroid *of the pupil threshold scan* in the *EyeSpace* window, regardless of what subsequent processing options are selected (c.f. [VPX_GetPupilPoint2](#).)

Note that this may not be equal to the value from [VPX_GetPupilPoint](#) that is the center of the fit ellipse.

Returns 1.

```
VPX_RealPoint pc;
VPX_GetPupilCentroid2 ( EYE_A, &pc );
printf( " X: %g , Y: %g ", pc.x, pc.y );
```

21.5.8 GetDiffVector

```
int VPX_GetDiffVector ( VPX_RealPoint *dv ); // EYE_A
int VPX_GetDiffVector2 ( VPX_EyeType eyn, VPX_RealPoint *dv );
```

Retrieves the **raw** normalized vector difference between the centers of the pupil and the glint in the *EyeSpace*.

Note: 0.5 is added to the values so that they are relative to the normalized center; the value is calculated as:

```
vector.x = ( pupilCenter.x - glintCenter.x ) + 0.5 ;
vector.y = ( pupilCenter.y - glintCenter.y ) + 0.5 ;
```

Returns 1.

```
VPX_RealPoint dv;
VPX_GetDiffVector2 ( EYE_A, &dv );
printf( " X: %g , Y: %g ", dv.x, dv.y );
```

21.5.9 GetGlintPoint

```
int VPX_GetGlintPoint ( VPX_RealPoint *gp ); // EYE_A
int VPX_GetGlintPoint2 ( VPX_EyeType eyn, VPX_RealPoint *gp );
```

Retrieves the **raw** normalized (x,y) location of the center of the glint (center of the OvalFit to the glint) in the *EyeSpace*.

Returns 1.

c.f. [VPX_GetGlintCentroid2](#)

```
VPX_RealPoint gp;
VPX_GetGlintPoint2 ( EYE_A, &gp );
printf( " X: %g , Y: %g ", gp.x, gp.y );
```

21.5.10 GetGlintCentroid

```
Int VPX_GetGlintCentroid2 ( VPX_EyeType eyn, VPX_RealPoint *gc );
```

Retrieves the **raw** normalized centroid of the glint threshold scan in the *EyeSpace* window, regardless of what subsequent processing options are selected (c.f. [VPX_GetGlintPoint2](#)).

Returns 1.

```
VPX_RealPoint gc;
VPX_GetGlintCentroid2 ( EYE_A, &gc );
printf( " X: %g , Y: %g ", gc.x, gc.y );
```

21.5.11 GetTorsion

```
int VPX_GetTorsion ( double *degrees ); // EYE_A
int VPX_GetTorsion2 ( VPX_EyeType eyn, double *degrees );
```

Retrieves torsion in +/- degrees.

The torsion measurement must be turned ON; it is off by default to save processing time.

3DWorkSpace & 3DViewPoint products set these values when they play back a 3D data file, but they are set by *ViewPoint* when it is sending data to the 3D products.

Returns 1.

```
Double degrees;
VPX_GetTorsion2 ( EYE_A, &degrees );
printf( " Torsion: %g ", degrees );
```

21.5.12 Data Quality

```
int VPX_GetDataQuality ( VPX_DataQuality *quality ); // EYE_A
int VPX_GetDataQuality2 ( VPX_EyeType eyn, VPX_DataQuality *quality );
```

Retrieves a code indicating what, if any, errors occurred during image processing and during testing against various data criteria levels.

Compare result against defined constants :

```
VPX_QUALITY_PupilScanFailed
VPX_QUALITY_PupilFitFailed
VPX_QUALITY_PupilCriteriaFailed
VPX_QUALITY_PupilFallback
VPX_QUALITY_PupilOnlyIsGood
VPX_QUALITY_GlintIsGood
```

3DWorkSpace & 3DViewPoint products set these values when they play back a 3D data file, but they are set by *ViewPoint* when it is sending data to the 3D products.

Returns 1.

```
VPX_DataQuality theQualityCode;
VPX_GetDataQuality2 ( EYE_A, & theQualityCode );
if ( theQualityCode == VPX_QUALITY_GlintIsGood )
{ // Glint and Pupil are good
}
else if (theQualityCode <= VPX_QUALITY_PupilFallback )
{ // Pupil only is good
}
```

21.6 Time Stamps

21.6.1 GetDateTime

```
int VPX_GetDateTime ( double *tm ); // EYE_A
int VPX_GetDateTime2 ( VPX_EyeType eyn, double *tm );
```

Retrieves the high precision time, in seconds, when the video frame became available for the current DataPoint, before video image processing and other calculations were done.

This was modified in version 2.8.2.36. Previously this obtained the time that the data was stored to the DLL and a `VPX_DAT_FRESH` event was issued. Now this function obtains the video synch time that better reflects the actual time that the image of the eye became available, and is not affected by variance in image processing time. The data storage time can now be obtained via `VPX_GetStoreTime2`.

Note: this modification affects `VPX_GetDataDeltaTime2` such that its variance should be significantly reduced.

Version 2.9.3.120 introduces `SINCE_SYSTEM_INIT_TIME`, but is currently used here.

Returns 1.

See also: `VPX_GetStoreTime2`

```
double tm;
VPX_GetDateTime2 ( EYE_A, &tm );
printf( " Time: %g ", tm );
```

21.6.2 GetDataDeltaTime

```
int VPX_GetDataDeltaTime ( double *tm ); // EYE_A
int VPX_GetDataDeltaTime2 ( VPX_EyeType eyn, double *tm );
```

Retrieves the high precision time interval, in seconds, between the last two `VPX_GetDateTime2` values.

Note: The delta time is affected by a change in `VPX_GetDateTime2` such that the variance of the delta times should be much less (since version 2.8.2.36).

Returns 1.

```
double tm;
VPX.GetDataDeltaTime2 ( EYE_A, &tm );
printf( " Time: %g ", tm );
```

21.6.3 GetStoreTime

```
int VPX_GetStoreTime2 ( VPX_EyeType eyn, double *tm );
```

Retrieves the high precision timestamp in seconds of the last time the data was stored to the DLL and a `VPX_DAT_FRESH` event was issued.

Use `VPX_GetDataTime` for eye movement times; use this to find out when the data was available in the DLL, e.g., to calculate the delay in inter-application notification event handling.

Version 2.9.3.120 introduces `SINCE_SYSTEM_INIT_TIME`, but is currently used here.

Returns 1.

```
double tm;
VPX_GetStoreTime2 ( EYE_A, &tm );
printf( " Time: %g ", tm );
```

21.6.4 GetStoreDeltaTime

```
int VPX_GetStoreDeltaTime2 ( VPX_EyeType eyn, double *tm );
```

Retrieves the high precision time interval, in seconds, between the last two `VPX_GetStoreTime2` values.

Returns 1.

```
double tm;
VPX_GetStoreDeltaTime2 ( EYE_A, &tm );
printf( " Time: %g ", tm );
```

21.6.5 Precision Timing

```
double VPX_GetViewPointSeconds
double VPX_GetPrecisionDeltaTime( double *pdInHoldSeconds, int iFlags );
```

Returns the time in SECONDS since `*pdInHoldSeconds`, or if `NULL`, since initialization.
 Returns `PDT_UNAVAILABLE` if the HighPerformanceCounter is NOT available.
 Returns `PDT_INITIALIZING` the first time, indicating initialization of the DLL initialization time variable.

In general, it is best to let the *ViewPoint* application start the DLL.

This function will automatically update the `*pdInHoldSeconds` with the current time, for use with the next call, if `iFlags` is `RESET_PRECISE_HOLD_TIME`, otherwise it will leave it unchanged. The four combinations are shown below:

28-Feb-2003

CHANGED behavior with `NULL` so that it returns the time since the first call to `VPX_GetPrecisionDeltaTime`.

Version 2.9.3.120 introduced the flag `SINCE_SYSTEM_INIT_TIME` and the second parameter was changed from `BOOL bResetHoldTime` to `INT iFlags`. All changes should be backwardly compatible and `SINCE_DLL_INIT_TIME` was added for clarity. Also, `inHoldTime` changed to `pdInHoldSeconds`, because it is now in *seconds*, which provides the user with a useful value; previously the value was in *clockTicks* that had not been converted to seconds.

See [20.6 High Precision Timing section](#).

```
#define VPX_GetViewPointSeconds \
    VPX_GetPrecisionDeltaTime( SINCE_PRECISE_INIT_TIME, SINCE_DLL_INIT_TIME );

static double someEventTime = 0.0;

secondsSinceLastSomeEvent =
    VPX_GetPrecisionDeltaTime( &someEventTime, RESET_PRECISE_HOLD_TIME );

secondsSinceLaunchingTheApp =
    VPX_GetPrecisionDeltaTime( &appLaunchTime, LEAVE_PRECISE_HOLD_TIME );

secondsSinceStartingTheDLL =
    VPX_GetPrecisionDeltaTime( SINCE_PRECISE_INIT_TIME, SINCE_DLL_INIT_TIME );

secondsSinceSystemWasBooted =
    VPX_GetPrecisionDeltaTime( SINCE_PRECISE_INIT_TIME, SINCE_SYSTEM_INIT_TIME );
```

21.7 Miscellaneous

21.7.1 GetCursorPosition

```
POINT VPX_GetCursorPosition ( void )
```

Returns the integer screen coordinates in pixels of the mouse cursor (pointer) on the machine that the DLL is running on.

Consequently, the return value will be different on the *ViewPoint* machine and a remote machine.

See also: [cursor_Control](#), [VPX_GetMeasuredScreenSize](#)

```
//Convert from pixel position to normalized screen position.  
POINT pixelCursorPosition = VPX_GetCursorPosition();  
VPX_RealPoint screenSize, normalizeCursorPosition, np;  
VPX_GetMeasuredScreenSize ( &screenSize ); // MUST be set correctly by user.  
normalizeCursorPosition.x = (float) pixelCursorPosition.x / screenSize.x ;  
normalizeCursorPosition.y = (float) pixelCursorPosition.y / screenSize.y ;  
  
//Put remoteMachine cursorPosition into ViewPoint DataFile.  
np = normalizeCursorPosition ;  
VPX_SendCommand(" dataFile_InsertString '\t%g\t%g' ", np.x, np.y );
```

21.7.2 GetStatus

```
int VPX_GetStatus ( VPX_StatusItem statusRequest )
```

This provides a simple way to determine the current status or state of various *ViewPoint* operations.

Note: The return value may need to be type cast for correct interpretation.

See also: `VPX_StatusItem`, `VPX_STATUS_CHANGE`, `status_Dump`

```
int running = VPX_GetStatus( VPX_STATUS_ViewPointIsRunning );
// regardless of whether ViewPoint or ViewPointClient is the local distributor.
int frozen = VPX_GetStatus( VPX_STATUS_VideoIsFrozen );
int open   = VPX_GetStatus( VPX_STATUS_DataFileIsOpen );
int paused = VPX_GetStatus( VPX_STATUS_DataFileIsPaused );
int thresh = VPX_GetStatus( VPX_STATUS_AutoThresholdInProgress );
// returns '0'=neitherEye thresholding, '1'=EyeA in process of thresholding,
// '2'=EyeB in process of thresholding, '3'=both eyes thresholding
int calib   = VPX_GetStatus( VPX_STATUS_CalibrationInProgress );
int binoc   = VPX_GetStatus( VPX_STATUS_BinocularModeActive );
int scene   = VPX_GetStatus( VPX_STATUS_SceneVideoActive );
char shape = (char)VPX_GetStatus( VPX_STATUS_StimulusImageShape );
// returns 'I'=isotropic stretch, 'C'=centered, 'F'=fit to window, 'A'=actual
int dllDataSource = VPX_GetStatus( VPX_STATUS_DistributorAttached );
// returns: 0=VPX_Distributor_None, 1=VPX_Distributor_IsViewPoint, or
// 2=VPX_Distributor_IsRemoteLink, 3=VPX_Distributor_IsEtherClient
```

21.7.3 GetViewPointHomeFolder

```
char* VPX_GetViewPointHomeFolder ( char* pathString );
```

Concatenates the full path to the *ViewPoint* folder onto the end of the provided string.

Must use 8-bit ASCII only.

Note: this is not a copy operation, i.e., it does not clear any existing contents of the provided string. To effectively obtain a copy operation, make sure an empty string is provided.

```
char pictureFile[512] = "" ;           // clear VERY IMPORTANT TO DO
VPX_GetViewPointHomeFolder( pictureFile ); // adds: ...ViewPoint/
lstrcat( pictureFile, IMAGE_FOLDER );    // adds "/Images/"
lstrcat( pictureFile, "myPicture.bmp" ); // adds "myPicture.bmp"
```

21.7.4 GetMeasuredViewingDistance

```
int VPX_GetMeasuredViewingDistance ( VPX_RealType *vd );
```

Provides the physical distance, in millimeters, of the subject's eye to the display screen, as specified by the user in the *ViewPoint Geometry window, 2D tab dialog*.

Note: Not valid when used with *3DWorkSpace* or *3DViewPoint*.

Returns 1.

See also:

[VPX_GetHeadPositionAngle](#)
[VPX_GetMeasuredScreenSize](#)

```
VPX_RealType vd;
VPX_GetMeasuedViewingDistance( &vd );
printf("Stimulus Display Viewing Distance is %g", vd );
```

21.7.5 GetMeasuredScreenSize

```
int VPX_GetMeasuredScreenSize ( VPX_RealPoint *sz );
```

Provides the physical size of the display screen, in millimeters, as calculated from the *ViewPoint Geometry window, 2D tab dialog*.

Note: The units entered in by the user in the *Geometry window* can be any length unit, and thus, all calculations and functions that deal with these units will remain in the user-specified units. The user MUST enter the viewing distance and vertical and horizontal line lengths.

Returns 1.

See also:

[VPX_GetMeasuredViewingDistance](#)

```
VPX_RealPoint sz ;
VPX_GetMeasuredScreenSize( &sz );
printf("Stimulus Display Screen Size is X: %g Y: %g", sz.x, sz.y );
```

21.8 Calibration Information

Get the `VPX_CalibrationEventRecord` immediately after receiving any calibration event (`VPX_CAL_*`). Refer to section [21.14.2](#) for a description of the calibration events and their sequencing.

21.8.1 VPX_CalibrationEventRecord

```
typedef struct {
    int calEvent;
    // VPX_CAL_WARN=5, VPX_CAL_BEGIN=6, VPX_CAL_SHOW=7, VPX_CAL_ZOOM=8,
    // VPX_CAL_SNAP=9, VPX_CAL_HIDE=10, VPX_CAL_END=11
    int index1; // cv.EyeSpace.selection1 { 1 ... N }
    VPX_RealPoint stimPtA, stimPtB;
    // calibration StimulusPoint location for EyeA and EyeB
    int slipMode; // boolean :: pv.doingSlipFix
    int snapMode; // boolean :: vpCv.calibration.snapMode
    // Used for timing : Snap OR Zoom : cv.calibration.snapMode
    // VPX_CAL_SNAP only, otherwise VPX_CAL_(BEGIN/ZOOM/SNAP/END)
    // from TIMER with ZOOM stimulus(i), calibratePoints1(a,a)
    int singlePoint; // boolean
    // True indicates that only a single point is being calibrated.
    // :: if True the range is calibratePoints1(a,a), otherwise calibratePoints1(a,b)
    int zoomSize; // N ... 0 ... N
    int eyes; // 1=EyeA, 2=EyeB, 3=Both
    // Eyes can be calibrated together or separately.
} VPX_CalibrationEventRecord;
```

21.8.2 GetCalibrationEventRecord

```
int VPX_GetCalibrationEventRecord( VPX_CalibrationEventRecord *calEventRec );
```

Retrieves the calibration event record for the current calibration event.

This function is designed to be used with the `VPX_CAL_*` members of the `VP_Message_NotificationCodes`. The `VPX_CalibrationEventRecord` contains the calibration stimulus point locations for both `EYE_A` and `EYE_B`, thus allowing different point locations between the two eyes.

Returns 0 on success, non-zero otherwise.

See also:

`VPX_GetCalibrationStimulusPoint2`
`VPX_CalibrationEventRecord`
`VP_Message_NotificationCodes`

```
VPX_CalibrationEventRecord calEventRec;
VPX_GetCalibrationEventRecord ( &calEventRec );
```

21.8.3 GetCalibrationStimulusPoint

```
int VPX_GetCalibrationStimulusPoint( int ixPt, VPX_RealPoint *calPt ); // EYE_A  
int VPX_GetCalibrationStimulusPoint2( VPX_EyeType eyn, int ixPt, VPX_RealPoint *calPt );
```

Retrieves the calibration *Stimulus Point* location for the ixPt point, in normalized units.

Valid ixPt range is {1 to 72}.

Returns 0 on success, non-zero otherwise.

See also: [VPX_GetCalibrationEventRecord](#)

CLI: [calibration_CustomPoint index xLoc yLoc](#)

CLI: [calibration_Snap](#)

```
VPX_RealPoint calPt;  
VPX_GetCalibrationStimulusPoint2 ( EYE_A, 1, &calPt );  
printf("CalPt X: %g Y: %g", calPt.x, calPt.y );  
MyCalibrationStimulusPointDraw(calPt);  
VPX_SendCommand("calibration_Snap");
```



21.9 HWND Functions

These are specific to Microsoft Windows machines and only work for layered applications that use the same DLL as *ViewPoint* is using, with the exception of `VPX_SetEyeImageWindow` that work via *ViewPointClient* on some recent version.

Note that the HWND functions may not be continued in future versions.

21.9.1 Set Remote EyeImage

```
int VPX_SetEyeImageWindow ( VPX_EyeType eyn, HWND hWnd );
```

Specifies the window within a layered application that should be used for display of the *EyeCamera* image.

In some later versions this works on remote MSWindows machines when video buffering is enabled.
Returns 1.

See also: `VPX_EyeCameraImageOverlays`; `VPX_SetEyeImageDisplayRect`

```
HWND hWnd = myEyeWindow ;  
VPX_SetEyeImageWindow( EYE_A, hWnd );
```

21.9.2 Set EyeImage Display Rectangle

```
int VPX_SetEyeImageDisplayRect ( VPX_EyeType eyn, RECT displayRect );
```

Allows optional re-adjustment of the display (*EyeCamera*) image offset and size, from the default.

Note: 320x240 provides optimal performance; other sizes may increase CPU usage.

Returns 1.

See also: `VPX_SetEyeImageWindow`

```
RECT displayArea = { 10, 10, 130, 250 } ;  
VPX_SetEyeImageDisplayRect( EYE_A, displayArea );
```

21.9.3 Set External Stimulus Window

```
int VPX_SetExternalStimulusWindow ( HWND hWnd );
```

This provides a mechanism for *ViewPoint* to draw the calibration stimuli directly into a window created by another application. The argument to this function is the handle of the created window.

CAUTION: When finished, the programmer must make certain to call this function again with a NULL HWND argument (or the handle of yet another window), before destroying the created window.

The calibration stimuli are still drawn in the *ViewPoint Stimulus* and *GazeSpace* windows, as usual.

Returns 1.

See sample code in: [VPX_MFC_Demo.cpp](#)

Contrast to: [HWND VPX_GetViewPointStimulusWindow \(\) ;](#)

```
if (1){ // WIN32
    HWND hWnd = myStimulusWindow ;
    VPX_SetExternalStimulusWindow( hWnd );
} else { // MFC
    CWnd* pWnd = GetDlgItem(IDC_StimulusPicture) ;
    HWND hWnd = pWnd->GetSafeHwnd() ;
    VPX_SetExternalStimulusWindow( hWnd );
}
```

21.9.4 *GazeSpace / Stimulus Window*

```
HWND VPX_GetViewPointStimulusWindow ( void );
HWND VPX_GetViewPointGazeSpaceWindow ( void );
```

Allows a layered program to access to *ViewPoint's Stimulus window* and *GazeSpace window* by way of the window handle.

Note: this will work only if layered application uses the same DLL that *ViewPoint* is using (i.e., the local data distributor application is *ViewPoint*. This does NOT work from remote computers, eg. using Ethernet!

See also:

```
VPX_SetEyeImageWindow
VPX_SetExternalStimulusWindow
VPX_GetStatus( VPX_STATUS_DistributorAttached );
```

```
int dllSource = VPX_GetStatus( VPX_STATUS_DistributorAttached );
if ( VPX_Distributor_IsViewPoint == dllSource )
{
    for ( int ix=0; ix<2; ix++ )
    {
        HWND hWnd ;
        if ( ix == 0 ) hWnd = VPX_GetViewPointStimulusWindow();
        else hWnd = VPX_GetViewPointGazeSpaceWindow();
        if (!hWnd) continue ;
        CWnd* w = FromHandle( hWnd );
        CDC* d = w->GetDC();
        RECT r ;
        w->GetClientRect(&r);
        d->Rectangle( r.left+10, r.top+10, r.right-10, r.bottom-10 );
        d->MoveTo( r.left, r.bottom ); d->LineTo( r.right, r.top );
        r.left = r.top = 25 ;
        d->DrawText("Drawing from\nVPX_mfc_demo", &r, 0);
        ReleaseDC(d);
    }
}
```

21.10 CallbackFunction Interface

Layered applications can access data and state changes from *ViewPoint* in real-time by registering one or more callback functions. A single callback function is usually sufficient. Note that if the callback function list becomes long, there may be a measurable time difference between the first and last.

The DLL creates a separate thread for each callback function. All callback functions are called upon an event and are processed in parallel according to the OS thread time slices. Once a function is called back, it will not be called again (interrupted) with a new event until the current function returns. You should return from a callback function as fast as possible preferably before the next event otherwise the Windows OS message queue may overrun. If using the data fresh events, you can choose to use the thinned operation flag that usually runs at ~30Hz regardless of the actual frame rate.

21.10.1 Insert Callback function

```
int VPX_InsertCallback ( VPX_CALLBACK callbackFunction, void* userPtr );
```

Inserts the specified `callbackFunction` into the list of callback functions that are called back, for example, when fresh data has been put in the DLL shared memory.

The `userPtr` is available to the user to specify any pointer that will later be returned (untouched) on the callback. Specifically it can be used to pass “`this`” within a Class, such that the `callbackFunction` that is outside the Class, can obtain a pointer to that Class. The way the `callbackFunction` can access the member functions of that class.

Returns: `VPX_CallbackResult`

See also: `VPX_RemoveNonRespondingMessageTargets`

Change Notes:

The `userPtr` parameter was added in version **2.9.FixMe**; previous versions did not contain this.

```
Int ret = VPX_InsertCallback( &myCallbackFunction, this );
```

21.10.2 Remove Callback function

```
int VPX_RemoveCallback ( VPX_CALLBACK callBackFunction );
```

Removes your application's request for notification for the specified `callBackFunction` that was created by `VPX_InsertCallback`. (not by the deprecated `VPX_InsertMessageRequest`)

Returns: `VPX_CallbackResult`

```
int ret = VPX_RemoveCallback( &myCallbackFunctionName );
```

21.10.3 List Callback functions

```
int VPX_GetCallbackListLength ( void );
```

Returns the number of callback functions that are registered by `VPX_InsertCallback` to receive function callbacks in the current process. (Note not by `VPX_InsertMessageRequest`)

Returns: The number of registered callback functions.

See also: `VPX_InsertCallback`

```
int nCallbacks = VPX_GetCallbackListLength();
```

21.10.4 Get Layered App CallbackListLength

```
int VPX_GetLayeredAppCallbackListLength ( void );
```

Returns the number of layered applications that have registered 1 or more event notifications (includes both callback functions and HWNDs).

See also:

`VPX_InsertCallback`

`VPX_InsertMessageRequest`

```
int ret = VPX_GetLayeredAppCallbackListLength();
```



21.10.5 VPX_CallbackResult

Check the return results to assure that the operation completed successfully.

```
typedef enum {
    VPX_CALLBACK_HEAD,
    VPX_CALLBACK_INSERTED,
    VPX_CALLBACK_INSERT_DUPLICATE,
    VPX_CALLBACK_LIST_LENGTH_EXCEEDED,
    VPX_CALLBACK_INSERT_ERROR,
    VPX_CALLBACK_REMOVED,
    VPX_CALLBACK_NOT_FOUND,
    VPX_CALLBACK_LIST_IS_EMPTY,
    VPX_CALLBACK_REMOVE_ERROR,
    VPX_CALLBACK_TAIL
} VPX_CallbackResult;
```

21.11 MessageRequest Interface (Deprecated)

DEPRECATED – Use the *CallBackFunction Interface* instead.

Layered applications can access data and state changes from *ViewPoint* in real-time by registering one or more message requests. Normally a single message request is sufficient. The messaging scheme was the original real-time interface developed in *ViewPoint* in the early 1990s. The platform independent Callback function interface is generally preferred now and future versions may not support the messageRequest scheme.

21.11.1 Insert MessageRequest

```
int VPX_InsertMessageRequest ( HWND hWnd, UINT msg )
```

Inserts the specified **hWnd** into the list of windows that are sent notification messages, e.g., when fresh data has been put in the DLL shared memory. Note that this causes messages to be sent, not CallBackFunction calls.

See also:

[VPX_InsertCallback](#)
[VPX_RemoveMessageRequest](#)
[VPX_RemoveNonRespondingMessageTargets](#)

```
int ret = VPX_RemoveNonRespondingMessageTargets( );
```

21.11.2 Remove MessageRequest

```
int VPX_RemoveMessageRequest( HWND hWnd );
```

Removes your application's request for notification for the specified window. Note that these entries were created by **VPX_InsertMessageRequest**, not by **VPX_InsertCallback**.

Returns ?

See also:

[VPX_InsertMessageRequest](#)
[VPX_RemoveCallback](#)

```
int ret = VPX_RemoveNonRespondingMessageTargets( );
```

21.11.3 Remove non-responding MessageRequests

```
int VPX_RemoveNonRespondingMessageTargets (void)
```

Removes all the non-responding message requests from message list. Note that these entries were created by **VPX_InsertMessageRequest**, not by **VPX_InsertCallback**.

Returns the number of non-responding message requests that were removed.

See also:

VPX_InsertCallback

```
int ret = VPX_RemoveNonRespondingMessageTargets( );
```

21.11.4 Get Message List Length

```
int VPX_GetMessageListLength ( int *num )
```

Returns the number of windows that are registered to receive messages in the current process. Note that these entries were created by **VPX_InsertMessageRequest**, not by **VPX_InsertCallback**.

See also:

VPX_InsertCallback

```
int ret = VPX_GetMessageListLength( &theMessageListLength );
```

21.11.5 Get Message Post Count

```
int VPX_GetMessagePostCount ( int *num )
```

Returns the total number of messages that have been distributed for all processes.

???? Note that these entries were created by **VPX_InsertMessageRequest**, not by **VPX_InsertCallback**. ????

See also:

VPX_InsertMessageRequest

```
int ret = VPX_GetMessagePostCount( &theNumberOfPosts );
```

21.11.6 GetViewPointAppCount

```
int VPX_GetViewPointAppCount (int *apps)
```

DEPRECATED

Sets applications to non-zero if *ViewPoint* is running.

See also:

```
int running = VPX_GetStatus( VPX_STATUS_ViewPointIsRunning );
```

```
int ret = VPX_GetViewPointAppCount( &vpApps );
```

21.12 Version Checking and Matching for SDK / DLL

When dynamically linked libraries (**DLL**) are created two files are created, one is a `.lib` file and the other is a `.dll` file on *Microsoft Windows* machines (`.dylib`, or `.so`, on *Mac* and *Linux* of machines, respectively). Programs are compiled with the `.lib` file, then when the program is run it dynamically links in the `.dll` file.

When layered applications communicate with *ViewPoint* on the same machine directly through a DLL, the version numbers must match for all layers:

- **vpx.h** header file for `.lib`, `.dll`, and all `.exe` files.
- **VPX_InterApp.lib** that was compiled into both *ViewPoint* and the *LayeredApp*.
- **ViewPoint.exe** the *ViewPoint* application.
- **VPX_InterApp.dll** linked at runtime into both *ViewPoint* and the layered app.

When layered applications communicate with *ViewPoint* through a *ViewPoint Client*, the following version numbers must all match:

- **vpx.h** header file for `.lib`, `.dll`, and all `.exe` files.
- **VPX_InterApp.lib** compiled into *ViewPoint*, *ViewPointClient* & *LayeredApp*.
- **ViewPoint.exe** the *ViewPoint* application.
- **VPX_InterApp.dll** linked at runtime into *ViewPoint*.
-
- **ViewPointClient.exe** client that communicates with the server in *ViewPoint*.
- **LayeredApp.exe** the user application
- **VPX_InterApp.dll** linked at runtime into *ViewPointClient* and the *LayeredApp*.

21.12.1 VPX_GetDLLVersion

```
double VPX_GetDLLVersion ( void );
```

Returns the version number of the loaded DLL.

The format of the returned number is: **ABC.xyz**, where the *ViewPoint* application would show the version as: **A.B.C.xyz**

See also:

[VPX_VersionMismatch](#)
[VPX_GetRevisionNumber](#)

```
double dllVersion = VPX_GetDLLVersion();
if ( VPX_SDK_VERSION != dllVersion ) doSomething();
if ( dllVersion is < 291.0 ) then doSomeOtherThing(); // version 2.9.1.000
```

21.12.2 VPX_VersionMismatch

```
BOOL VPX_VersionMismatch ( VPX_SDK_VERSION );
```

Returns 0 if the program was compiled with the same version of the DLL lib as the DLL that is loaded at run time.

Version 2.9.3.115 and after only check the first three values of the version quad, the last value is for small patches that do not effect interapplication communication.

See also:

[VPX_GetDLLVersion](#)

```
If ( VPX_VersionMismatch( VPX_SDK_VERSION ) ) return;
```

21.12.3 VPX_GetRevisionNumber

```
double VPX_GetRevisionNumber( void ); // DEPRECATED
```

Returns small revisions to DLL, mostly used for DLL development.

See also:

[VPX_GetDLLVersion](#)
[VPX_VersionMismatch](#)

```
int ret = VPX_GetRevisionNumber();
```

21.13 SDK Utility Functions

21.13.1 DebugSDK

```
int VPX_DebugSDK( BOOL flag );
```

If TRUE the SDK interface will provide more information that can help with debugging.

```
VPX_DebugSDK(1);
```

21.13.2 Draw Rectangle

```
VPX_RectFrame ( HDC hdc, int x1, int y1, int x2, int y2, int t );
```

Draws two concentric hollow rectangles in the specified window. The inner rectangle is defined by the specified coordinates. The outer rectangle is larger by parameter t pixels.

```
VPX_RectFrame ( hDC, r.left, r.top, r.right, r.bottom, nBigger );
```

21.13.3 Draw Ellipse

```
VPX_EllipseFrame ( HDC hdc, int x1, int y1, int x2, int y2, int t );
```

Draws two concentric hollow ellipses in the specified window. The inner rectangle is defined by the specified coordinates. The outer rectangle is larger by parameter t pixels.

```
VPX_EllipseFrame ( hDC, r.left, r.top, r.right, r.bottom, nBigger );
```

21.13.4 Draw ROI

```
VPX_drawROI ( HWND hWnd, int activeRegion );
```

Draws the activeRegion ROI in red and all of the other ROI in blue, within the specified window.

```
VPX_drawROI ( hWnd, activeRegion );
```

21.13.5 Convert: pixelRect → normalizedRect

```
VPX_WindowRECT2RealRect ( RECT nr, RECT clientRect, RealRect * rr );
```

Takes in integer coordinates for a rectangle within a specified window and returns the normalized coordinates for that rectangle. This is useful for obtaining a *normalizedRect* from a *windowsRect* that is in a particular *clientRect*

```
RECT pixelRect = { 50, 50, 99, 99 };
VPX_RealRect normalizedRect;
VPX_WindowRECT2RealRect ( pixelRect, clientRect, & normalizedRect );
```

21.13.6 Convert: normalizedRect → pixelRect

```
VPX_RealRect2WindowRECT ( RealRect rr, RECT clientRect, RECT *scaledRect );
```

Takes normalized coordinates of a rectangle and returns integer (pixel) coordinates that have been scaled for the size of the specified window.

```
RealRect rr = { 0.1, 0.1, 0.2, 0.2 };
RECT clientWindowRect = { 320, 240 };
RECT scaledRect ;
VPX_RealRect2WindowRECT( rr, clientWindowRect, &scaledRect );
// The scaledRect will now contain { 32, 24, 64, 48 }
```

21.13.7 Convert: LParam → RealPoint

```
int VPX_LParam2RealPoint ( LPARAM codedLoc, VPX_RealPoint *pt );
```

Legacy MessageRequest interface.



21.13.8 Convert: LParam → RectPoint

```
int VPX_LParam2RectPoint ( LPARAM codedLoc, RECT clientRect, POINT *pt );
```

Legacy MessageRequest interface.

Used with VPX_CAL_* messages to obtain the location of the calibration point that is encoded in the message LPARAM. Takes LPARAM and returns integer coordinates of the calibration points that have been scaled for the size of the specified window. Previously defined in DLL but not listed in prototypes, because it was under evaluation. Added here in version 2.4.2.0

See also: [VPX_GetCalibrationStimulusPoint](#)

21.14 Events and Notification Messages

ViewPoint is constantly sending messages to layered applications notifying them that there is fresh data available, that a *DataFile* has been opened/paused/resumed/closed, etc. These messages can be obtained in real-time via a CallBackFunction that is activated by `VPX_InsertCallback` (or by the deprecated Windows messaging scheme that is activated by using `VPX_InsertMessageRequest`)

21.14.1 General Events

msg HIWORD(WPARAM)	<code>VPX_DAT_FRESH</code> The data has just been updated; real-time programs should now access the data that it needs by calling the accessor functions.
subMsg LOWORD(WPARAM)	The eye the command pertains to: <code>EYE_A</code> , <code>EYE_B</code>
param1 HIWORD(LPARAM)	Not used.
param2 LOWORD(LPARAM)	Not used.
<pre>VPX_EyeType eyn = (VPX_EyeType) subMsg; VPX_EyeType eyn = (VPX_EyeType) LOWORD(wparam); VPX_RealRect gpt; VPX_GetGazePoint2(eyn, &gpt);</pre>	



Arrington Research

msg HIWORD(WPARAM)	VPX_ROI_CHANGE Indicates that a Region Of Interest (ROI) was changed.
subMsg LOWORD(WPARAM)	<pre>RealRect rr ; RECT cr, dr ; GetClientRect(hwnd, &cr); WORD roiIndexNumber = LOWORD(wParam); VPX_GetROI_RealRect(roiIndexNumber, &rr); VPX_RealRect2WindowRECT(rr, cr, &dr); Rectangle(hdc, dr.left, dr.top, dr.right, dr.bottom);</pre>
LPARAM	Do not use LPARAM.

msg HIWORD(WPARAM)	VPX_STATUS_CHANGE Indicates that a key <i>ViewPoint</i> status item was changed. For details, see section 21.7.2: GetStatus . VPX_GetStatus
subMsg LOWORD(WPARAM)	Not used.
param1 HIWORD(LPARAM)	<pre>WORD statusValue = param1; WORD statusValue = HIWORD(lParam);</pre>
param2 LOWORD(LPARAM)	<pre>WORD statusItem = param2; WORD statusItem = HIWORD(lParam);</pre>
<pre>WORD statusItem = param1; WORD statusValue = param2; switch (statusItem) { case VPX_STATUS_DataFileIsOpen : printf("DataFile is %s", (statusValue==1)? "Open" : "Closed"); break; ... }</pre>	



Arrington Research

msg HIWORD(WPARAM)	<code>VPX_VIDEO_FrameAvailable</code> Notifies external (layered) applications that a video frame is available in <i>ViewPoint</i> memory. See also:
subMsg LOWORD(WPARAM)	<code>VPX_EyeType eye = LOWORD(wparam);</code> <code>// Usage: if (eye == EYE_A)</code>
LPARAM	<code>DWORD notUsed = lparam; // NOTE: subject to change!</code>

msg HIWORD(WPARAM)	<code>VPX_VIDEO_SyncSignal</code> This message is sent as soon as the video capture board detects frame-ready (30 Hz) or field-ready (60 Hz) signal. The user can now tell when the image became available for processing, before any image processing has been performed. This better reflects the true time of the eye movement, and reduces noise in the timing calculation. See also: <code>vpx_event +videoSynch</code> <code>VPX_GetDataTime2</code>
subMsg LOWORD(WPARAM)	<code>VPX_EyeType eye = LOWORD(wparam);</code> <code>if (eye == EYE_A) { /* do it */ }</code>
LPARAM	<code>DWORD deltaMicroSeconds = lparam;</code> <code>// NOTE: subject to change!</code>

21.14.2 Calibration Events

The flow of the calibration events in the autoCalibration sequence is as follows:

```

VPX_CAL_BEGIN
VPX_CAL_WARN
    // each calibration point
    VPX_CAL_SHOW
        VPX_CAL_ZOOM // for radius 15 down to 0
        VPX_CAL_SNAP
        VPX_CAL_ZOOM // for radius 0 up to 15
    VPX_CAL_HIDE
VPX_CAL_END

```

Call `VPX_GetCalibrationEventRecord` (see section [21.8.1](#)) immediately after receiving any calibration event.

msg HIWORD(WPARAM)	<code>VPX_CAL_BEGIN</code> Indicates that a calibration sequence is about to start. This is the first calibration message in the sequence. In general you would want to blank the stimulus display screen and disable other graphics drawing.
subMsg LOWORD(WPARAM)	The calibration point number, the actual point index number, not the random or custom sequence number.
LPARAM	The location of the upcoming StimulusPoint. As below, use: <code>VPX_LParam2RectPoint(lParam, cr, &calPt);</code>



msg HIWORD(WPARAM)	<p>VPX_CAL_WARN</p> <p>Provides an opportunity to display a warning message to the subject, to make sure that they are paying attention.</p> <p>Follows VPX_CAL_BEGIN.</p> <p>The warning time, i.e., the delay between this event and the next event, can be specified in <i>ViewPoint</i>, <i>EyeSpace</i> window, Advanced button, WarningTime slider.</p>
subMsg LOWORD(WPARAM)	The calibration point number, the actual point index number, not the random or custom sequence number.
LPARAM	<p>Contains the location of the upcoming StimulusPoint.</p> <pre>POINT calPt; char* str = " PAY ATTENTION " ; RECT cr ; GetClientRect(hwnd, &cr); VPX_LParam2RectPoint(lParam, cr, &calPt); TextOut(hdc, calPt.x-80,calPt.y, str, strlen(str));</pre>

msg HIWORD(WPARAM)	<p>VPX_CAL_SHOW</p> <p>Indicates that the calibration StimulusPoint should be drawn.</p> <p>Follows VPX_CAL_WARN for the first StimulusPoint; loops back to here after VPX_CAL_HIDE for each additional StimulusPoint.</p>
subMsg LOWORD(WPARAM)	The calibration point number, the actual point number, not the random or custom sequence number.
LPARAM	<p>Contains the location of the StimulusPoint.</p> <p>As above, use: VPX_LParam2RectPoint(lParam, cr, &calPt);</p>



msg HIWORD(WPARAM)	VPX_CAL_ZOOM Indicates a radius change of the tunnel motion of the stimulus. Follows VPX_CAL_SHOW and is repeatedly sent until the radius shrinks to zero.
subMsg LOWORD(WPARAM)	The stimulus radius (shrinks from 15 to 2). <pre>WORD zoomSize = subMsg; WORD zoomSize = LOWORD(wParam);</pre>
LPARAM	Contains the location of the StimulusPoint. <pre>POINT pt ; RECT cr ; GetClientRect(hwnd, &cr); VPX_LParam2RectPoint(lParam, cr, &pt); int scaleToWindowSize = cr.right / 200 ; r = zoomSize * scaleToWindowSize; // radius Rectangle (hdc, pt.x - r, pt.y - r, pt.x + r, pt.y + r);</pre>
	<pre>POINT pt ; int zoomSize = subMsg; RECT cr ; GetClientRect(hwnd, &cr); VPX_CalibrationEventRecord calEventRec; VPX_GetCalibrationEventRecord (&calEventRec); pt.x = calEventRec.stimPtA.x * cr.right; pt.y = calEventRec.stimPtA.y * cr.bottom; int scaleToWindowSize = cr.right / 200 ; int r = zoomSize * scaleToWindowSize ; // radius Rectangle (hdc, pt.x - r, pt.y - r, pt.x + r, pt.y + r);</pre>



msg HIWORD(WPARAM)	VPX_CAL_SNAP Indicates that the calibration image of the eye is being taken. Follows the series of VPX_CAL_ZOOM events, after zoomSize has shrunk to zero; or if the calibration mode is in snapMode, this is called itself.
subMsg LOWORD(WPARAM)	The calibration point number, the actual point index number, not the random or custom sequence number, is in the lower 8 bits, flags for slipCorrection mode and snapMode are in the upper 8 bits. <code>subMsg ???</code> <code>WORD loWordw = LOWORD(wParam);</code> <code>BOOL slipMode = (loWordw & 128) ? 1 : 0 ; // bit 8</code> <code>BOOL snapMode = (loWordw & 256) ? 1 : 0 ; // bit 9</code> <code>int pointNumber = LOWORD(wParam) & 127 ; // low bits 0..7</code>
LPARAM	Contains the location of the StimulusPoint. As above, use: <code>VPX_LParam2RectPoint(lParam, cr, &calPt);</code>

msg HIWORD(WPARAM)	VPX_CAL_HIDE Indicates completion of the current calibration point. The program should clean up any remnants of this last calibration StimulusPoint display.
subMsg LOWORD(WPARAM)	The calibration point number, the actual point index number, not the random or custom sequence number. <code>int pointNumber = LOWORD(wparam)</code>
LPARAM	Contains the location of the StimulusPoint. As above, use: <code>VPX_LParam2RectPoint(lParam, cr, &calPt);</code>



msg HIWORD(WPARAM)	VPX_CAL_END Indicates that the entire calibration sequence has finished. The LOWORD indicates whether or not a slipFix was requested. A 1 indicates slipFix, zero indicates (re)calibration of a point.
subMsg LOWORD(WPARAM)	Indicates whether or not a slipFix was requested (rather than e.g. a recalibration) . A 1 indicates slipFix, zero indicates recalibration of a point. Note: this is not currently consistent with the wparam format used in VPX_CAL_SNAP, but it may be made consistent in the future. BOOL doSlipFix = LOWORD(wParam) == 1 ;
LPARAM	Contains the location of the StimulusPoint. As above, use: VPX_LParam2RectPoint(IParam, cr, &calPt);

21.15 Structures and Enumerations

21.15.1 VPX_PositionAngle

```
typedef struct {
    VPX_RealType x;           // horizontal-axis
    VPX_RealType y;           // vertical-axis
    VPX_RealType z;           // depth-axis
    VPX_RealType roll;        // about the z-axis (depth-axis)
    VPX_RealType pitch;       // about the x-axis (horizontal-axis)
    VPX_RealType yaw;         // about the y-axis (vertical-axis)
} VPX_PositionAngle;
```

21.15.2 VPX_GetImageRecord

```
typedef struct {
    int sourceBuffer; // Specifies the source buffer to get the image from. (EYE_A, EYE_B, SCENE_A, SCENE_B)
    int destWidthStep; // Size of aligned (padded) image row in bytes. If no padding then destWidthStep = (width * bytesPerChannel * colorChannels).
    int destWidth; // Image width in pixels.
    int destHeight; // Image height in pixels
    VPX_IntRect destRect; // Currently not used.
    VPX_GetImageBitsPerChannel destBitsPerChannel; // Number of bits per channel.
    VPX_GetImageColorChannels destColorChannels; // Number of color channels.
    int flipAroundHorizontalAxis; // Flips the image about the horizontal axis (Flip Image Vertically). (nonzero - flip, 0 - no flip).
    int flipAroundVerticalAxis; // Flips the image about the vertical axis (Flip Image Horizontally). (nonzero - flip, 0 - no flip).
    int swapRedAndBlue; // Swaps the Red and Blue color values (converts BGR2RGB, RGB2BGR, BGRA2RGBA, RGBA2BGRA). (nonzero - swap, 0 - no swap).
    int setAlphaValue; // Sets all Alpha values with alphaValue (only valid if destColorChannels has a value of 4). (nonzero - set, 0 - no set).
    unsigned char alphaValue; // Value used to set all Alpha values when setAlphaValue is nonzero. (Valid range is 0 - 255).
} VPX_GetImageRecord;
```

21.15.3 VPX_QUALITY_

#define VPX_DataQuality int		
#define VPX_QUALITY_PupilScanFailed	5	// pupil scan threshold failed.
#define VPX_QUALITY_PupilFitFailed	4	// pupil could not be fit with an ellipse.
#define VPX_QUALITY_PupilCriteriaFailed	3	// pupil was bad because it exceeded criteria limits.
#define VPX_QUALITY_PupilFallBack	2	// wanted glint, but it was bad, using the good pupil.
#define VPX_QUALITY_PupilOnlyIsGood	1	// wanted only the pupil and got a good one.
#define VPX_QUALITY_GlintIsGood	0	// glint and pupil are good.

21.15.4 VPX_GLINT_QUALITY

#define VPX_GlintDataQuality int		
#define VPX_GLINT_QUALITY_ScanFailed	5	// glint scan threshold failed.
#define VPX_GLINT_QUALITY_FitFailed	4	// glint could not be fit.
#define VPX_GLINT_QUALITY_WidthCriteriaFailed	3	// glint was bad because it exceeded width criteria limits.
#define VPX_GLINT_QUALITY_AspectCriteriaFailed	2	// glint was bad because it exceeded aspect criteria limits.
#define VPX_GLINT_QUALITY_NoOperation	1	// glint skipped because pupil was not found.
#define VPX_GLINT_QUALITY_Good	0	// glint is good.

21.15.5 VPX_STATUS_

```

typedef enum {
    VPX_STATUS_HEAD=0,
    VPX_STATUS_ViewPointIsRunning,           // returns bool, if true, it may be running on remote machine
    VPX_STATUS_VideosFrozen,                // returns bool
    VPX_STATUS_DataFileIsOpen,              // returns bool
    VPX_STATUS_DataFileIsPaused,             // returns bool
    VPX_STATUS_AutoThresholdInProgress,     // returns bool
    VPX_STATUS_CalibrationInProgress,       // returns bool
    VPX_STATUS_StimulusImageShape,          // returns 'I'=isotropic stretch, 'C'=centered, 'F'=fit to window, 'A'=actual
    VPX_STATUS_BinocularModeActive,          // returns bool
    VPX_STATUS_SceneVideoActive,             // returns bool
    VPX_STATUS_DistributorAttached,         // return which VPX\_DistributorType is connected to this local DLL.
    VPX_STATUS_CalibrationPoints,           // TENTATIVE :: returns the number of calibration points: 6,9,12,...,72
    VPX_STATUS_TTL_InValues,                // TENTATIVE :: bit code for ttl hardware input channels
    VPX_STATUS_TTL_OutValues,               // TENTATIVE :: bit code for ttl hardware output channels
    VPX_STATUS_TorsionActive,               // returns 0 = None, 1 = EYE_A, 2 = EYE_B, 3 = EYE_A && EYE_B.
    VPX_STATUS_BinocularAveraging,           // returns VPX\_BinocularAveragingType, see section 21.15.6
    VPX_STATUS_TAIL                         // Use with: VPX\_GetStatus, eg. after VPX\_STATUS\_CHANGE notification
} VPX_StatusItem;

```

21.15.6 VPX_BinocularAveragingType

```

typedef enum {
    VPX_BinocularAveraging_Off,            // No averaging
    VPX_BinocularAveraging_Only_Y,          // Averages y for both eyes.
    VPX_BinocularAveraging_Both_XY,          // Averages x and y for both eyes to create a single gaze point.
    VPX_BinocularAveraging_ParallaxCorrection, // Same as Both_XY with parallax correction.
} VPX_BinocularAveragingType;

```

21.15.7 VPX_DistributorType

```

#define VPX_DistributorType      int
#define VPX_Distributor_None     0
#define VPX_Distributor_IsViewPoint 1
#define VPX_Distributor_IsRemoteLink 2
#define VPX_Distributor_IsEtherClient 3

```

The DLL based SDK gets data from, and sends command strings to, a "distributor" application. Normally the distributor application is the *ViewPoint EyeTracker*, but it could be *ViewPointClient* application.

// Note: [VPX_STATUS_ViewPointIsRunning](#) returns true if *ViewPoint* is running either directly or via *ViewPointClient*.

The *ViewPoint* distributor does two main things: (a) updates the data in the shared memory of the library, (b) makes sure that [VPX_SendCommand](#) instructions are sent to the *ViewPoint Command Line Interface* (CLI). Examples of distributor's applications are: the *ViewPoint* itself, the *ViewPointClient* application, for PC, the MAC dylib that contains the *ViewPointClient*.

```
VPX_DistributorType source = VPX_GetStatus( VPX_STATUS_DistributorAttached );
```

21.15.8 VP_Message_NotificationCodes

```

typedef enum {
    VPX_ENUM_NOTIFICATIONS_HEAD = 0,
    VPX_Obsolete_01 = 1,
    VPX_DAT_FRESH = 2, // there is fresh data available
    VPX_Obsolete_03 = 3,
    VPX_Obsolete_04 = 4,
    VPX_CAL_WARN = 5, // Added vp.2.8.1.12 , follows VPX_CAL_BEGIN to notify "GET READY"
    VPX_CAL_BEGIN = 6, // begin calibration sequence
    VPX_CAL_SHOW = 7, // index, (y,x)
    VPX_CAL_ZOOM = 8, // radius, (y,x)
    VPX_CAL_SNAP = 9, // index, (y,x)
    VPX_CAL_HIDE = 10, // index, (y,x)
    VPX_CAL_END = 11, // end calibration sequence, doingSlipFix=LOWORD
    VPX_CAL_TAIL = 12,
    VPX_ROI_CHANGE = 13, // Region Of Interest (ROI) was changed somewhere
    VPX_Obsolete_14 = 14,
    VPX_Obsolete_15 = 15,
    VPX_Obsolete_16 = 16,
    VPX_Obsolete_17 = 17,
    VPX_Obsolete_18 = 18,
    VPX_Obsolete_19 = 19,
    VPX_COMMAND_STRING = 20,
    VPX_STATUS_CHANGE = 21,
    VPX_Obsolete_22 = 22,
    VPX_Obsolete_23 = 23,
    VPX_Obsolete_24 = 24,

    VPX_VIDEO_FrameAvailable = 25, // was = 501, changed ViewPoint version 2.8.3.457.
        // Notifies layered apps that a new image has been updated in their remote window or
        // copied into the dll buffer. LOWORD(lParam) == 0 (remote image),
        // LOWORD(lParam) == 1 (dll buffer) when using CLI command "ImageBuffer +EyeA"

    VPX_TRIGGER_EVENT = 26, // was = 504, changed ViewPoint version 2.8.3.457
    VPX_VIDEO_SyncSignal = 27, // was = 505, changed ViewPoint version 2.8.3.457.
        // Earliest notification that a new video image is ready, before image processing and data

    VPX_VIDEO_BufferedImageAvailable = 28,
        // Buffered Image available in dll (lParam is buffer index).

    _DAT_FRESH_BufferedEyeDataAvailable = 29,
        // Buffered EyeData available in dll (lParam is buffer index).

    VPX_TTL_IN = 164, // Reserved: ViewPoint internal use only.
    VPX_TTL_OUT = 165, // Reserved: ViewPoint internal use only.
    VPX_ENUM_NOTIFICATIONS_TAIL
} VP_Message_NotificationCodes;

```

21.15.9 VPX_ParseType

The `VPX_ParseType` enums refer only to success or failure of the parsing operation. They do not refer to the success or failure of `ViewPoint` subsequently executing the command.

```
typedef enum {
    VPX_PARSE_HEAD=0,           // 0 should never occur
    VPX_PARSE_OK,              // 1 indicates OK : New 2.8.1.009, was (PARSE_COMMENT)
    VPX_PARSE_ACTION,          // 2 indicates immediate return
    VPX_PARSE_END,              // 3 ( result <= VPX_PARSE_END ) ? "OK" : "ERROR"
    VPX_PARSE_COMMENT,          // 4 <- INSERTED, moved from VPX_PARSE_OK location 2.8.1.009
    VPX_PARSE_OBSOLETE,         // OBSOLETE commands.
    VPX_PARSE_ERROR_HEAD,       // error = ( result > VPX_PARSE_ERROR_HEAD )
    VPX_PARSE_ERROR_UnknownCommand, // from: VP parser; Command not recognized by the parser.
    VPX_PARSE_ERROR_MissingParameter, // from: VP parser; Command is missing a parameter.
    VPX_PARSE_ERROR_EmptyLine,   // from: VP parser; Command is empty or whitespace. new 2.7.0.090
    VPX_PARSE_ERROR_SendMessageTimeOut, // sent from: DLL; Timeout occurred waiting for VP parser.
    VPX_PARSE_ERROR_IllegalParameter, // sent from: VP parser; Command has an illegal parameter.
    VPX_PARSE_ERROR_ParserIsNotRunning, // sent from: DLL; ViewPoint Parser is not running.
    VPX_PARSE_ERROR_SendMessageFailed, // sent from: DLL, Failed to communicate with VP parser.
    // (Try running all apps using the dll as administrator or not)
    VPX_PARSE_TAIL
} VPX_ParseType;
```

21.15.10 VPX_CallbackResult

```
typedef enum {
    VPX_CALLBACK_HEAD,
    VPX_CALLBACK_INSERTED,
    VPX_CALLBACK_INSERT_DUPLICATE,
    VPX_CALLBACK_LIST_LENGTH_EXCEEDED,
    VPX_CALLBACK_INSERT_ERROR,
    VPX_CALLBACK_REMOVED,
    VPX_CALLBACK_NOT_FOUND,
    VPX_CALLBACK_LIST_IS_EMPTY,
    VPX_CALLBACK_REMOVE_ERROR,
    VPX_CALLBACK_TAIL
} VPX_CallbackResult;
```

21.15.11 VPX_EyeDataRecord

```

typedef struct {
    double torsionDegrees ;
    double pupilAspectRatio ;
    double dataTime ;           // precision time value at last video sync time
    double dataDeltaTime ;      // difference between previous and current dataTime
    double storeTime ;          // precision time value at last fresh data
    double storeDeltaTime ;     // difference between previous and current dataTime
    double fixationSeconds ;    // new: precision seconds binocular
    double totalDrift ;
    double totalVelocity ;

    // GazeSpace Data
    VPX_RealPoint    gazePoint ;
    VPX_RealPoint    gazePointSmoothed ;
    VPX_RealPoint    gazePointCalculated ;
    VPX_RealPoint    gazeAngle ;
    VPX_RealPoint    gazeAngleSmoothed ;
    VPX_RealPoint    gazeAngleCalculated ;
    VPX_RealPoint    pupilSize ;
    double   pupilDiameter_MM;
    double   pupilAngle ;
    int      regionCode ;
    int      fixationDuration ; // old: milliseconds monocular deprecated
    VPX_EyeEventType blinkEvent ;
    VPX_EyeEventType moveEvent ;
    VPX_RealPoint    componentVelocity ;

    // EyeSpace Data
    VPX_RealPoint    pupilCentroid ;
    VPX_RealPoint    glintCentroid ;
    VPX_RealRect    pupilOvalRect ;
    VPX_RealPoint    pupilPosition ;
    VPX_RealPoint    glintPosition ;
    VPX_RealPoint    diffVector ;
    VPX_GlintRecord glintList[MAX_GLINTS];
    int      glintCount;      // Number of active glints in the glintList.
    int      dataQuality ; // Quality for entire EyeDataRecord (includes pupil and glints).
    int      panelHit ;
    VPX_RoiHitListType    roiHitList;    // The first time, Clear will set all to -1
    VPX_RoiHitListType    roiEventList; // +N indicates newly inside, -N indicates newly outside
    int      roiHitListLength;
    VPX_EyeType        eye;
} VPX_EyeDataRecord;

```

Chapter 22. Troubleshooting

This section discusses some of the common sources of error and problem areas. Once recognized, many of these can be avoided.

22.1 History Window

The *History* window can be a very useful tool for troubleshooting many problems including video and *Settings* file problems. Use menu item: [Windows > History](#) to show the window.

Selecting menu item: [File > Settings > Verbose Loading](#) will display extra information from the parser, which is useful for troubleshooting the CLI commands.

22.2 Improving Frame Rate

The video frame rate will be compromised when other demands are made on the computer. Ways to improve video frame rate include:

Closing the *EyeSpace* window and the *History* window will significantly improve performance.

Turn off “[Show Dots](#)” in the *EyeCamera* window.

Turn off the screen saver.

Ensure that other applications and background processes that are not required are not running.

22.3 EyeCameraWindow Troubleshooting

If a video source was connected to the computer when *ViewPoint* was started, the *EyeCamera* window should display the captured video image. Otherwise, follow the following troubleshooting tips:

The *EyeCamera* window shows [*** FROZEN ***](#), then you should select the snowflake icon on the *EyeCamera* window to unfreeze the video processing.

Ensure that the frame grabber board and drivers have been correctly installed. Check in the [Windows Device Manager](#) for conflicts.

Reset the video: [EyeCamera window > monitor icon > Reset EyeCamera Video](#)

Note: If a camera is disconnected and reconnected then *ViewPoint* will automatically try to reset the video pipe to get it working again. *ViewPoint* tries 2 resets and displays each attempt in the *History* window.

If the *EyeCamera* window background is black, white or blue, then check the following:

The camera is plugged into the computer properly

The camera is getting power that it needs, e.g., from a power supply

The camera is getting enough light

The camera iris adjustment is open

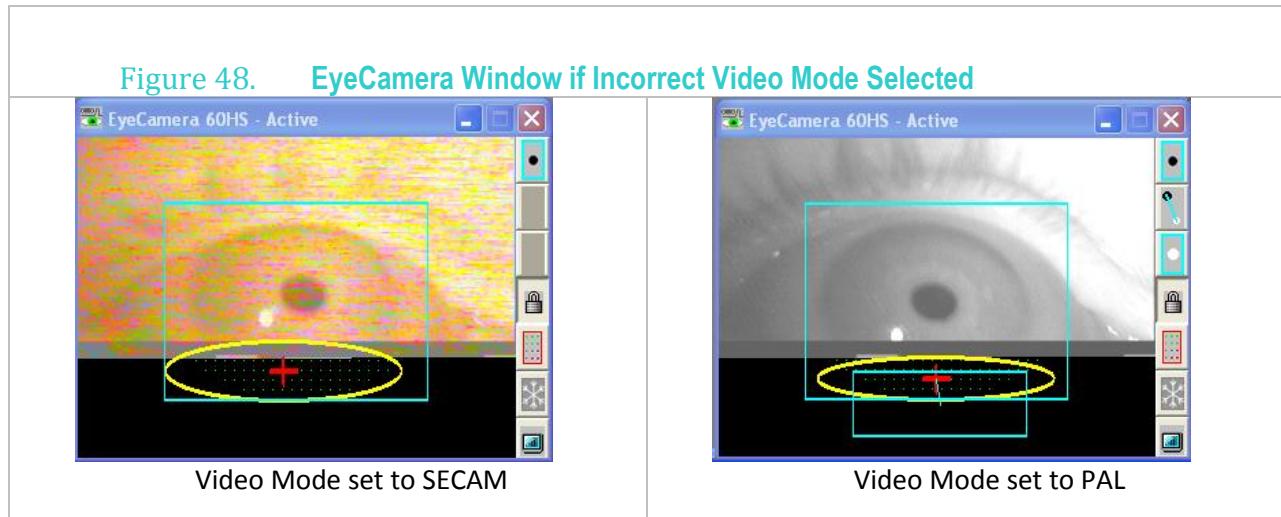
The lens cap has been removed!

If the *EyeCamera* window video segmentation is not working make sure the display monitor is set to True Color (32 bit).

22.3.1 Bottom Half of EyeCamera Window is Black (Analog 60 Hz products)

If the bottom of *EyeCamera* image is black as in [Figure 48](#) then the video standard has been set to PAL or SECAM, but the camera is NTSC. All ARI supplied cameras are NTSC. Select the monitor icon on

the *EyeCamera window > Video Standard > NTSC*. Also check any *Settings* file that may be loaded, e.g. *StartUp.txt* that may specify a different video standard.



High Speed ViewPoint (e.g. USB 220), may have a black band at the bottom as part of normal operation.

22.4 General Troubleshooting

If the color of the sliders and buttons are different than the window background, change the Microsoft Windows appearance settings in the *Windows Controls > Display > Appearance* to the *Windows Standard* scheme.

Chapter 23. Error Codes

23.1 Introduction to Error Codes

Check whether the error code has a description and remedy below. If not, *ViewPoint* error codes provide a good way for customers to precisely communicate problems to our support team, and thus more quickly obtain useful help.

Error #	Error String	Remarks & Remedies
0	VP_ERROR_Undefined	This is a place holder and should be used.
	Launch errors	
1	VP_ERROR_UnsupportedOS	
2	VP_ERROR_MaxLicenses	
3	VP_ERROR_DllsLocked	
4	VP_ERROR_AppCountTwoExceeded	
5	VP_ERROR_AppCountOneExceeded	
6	VP_ERROR_GetMessage_InvalidPointer	
7	VP_ERROR_SetWindowPosition	
8	VP_ERROR_SetWindowSize	
9	VP_ERROR_SetWindowtRec	
22	VP_ERROR_ElevatedVistaModeRequired	Run app as administrator OBSOLETE
120	VP_ERROR_ControlTabInit	
121	VP_ERROR_ControlTabNullWindow	
122	VP_ERROR_ControlTabInsertItem	
123	VP_ERROR_ControlTabIsNull	
130	VP_ERROR_ListBoxDeleteStringError	
140	VP_ERROR_LoadMenuFailed	
150	VP_ERROR_normalValue_fromSlider	
160	VP_ERROR_RevertRegion	
	200s file errors	
201	VP_ERROR_BadDataFileHandle	
202	VP_ERROR_FileNameIllegalChars	
203	VP_ERROR_FileNameEmptyString	
204	VP_ERROR_ImageFileLoad	
205	VP_ERROR_EyeMovieSetThreadPriority	
206	VP_ERROR_CannotOpenFile	
207	VP_ERROR_DataFile_NeverUnpaused	
220	VP_ERROR_StimulsDisplay	
221	VP_ERROR_HideStimulsWindowError	
	300s are PenPlot errors	
300	VP_ERROR_PenPlots	
301	VP_ERROR_PenPlot_HiLoMismatch	
302	VP_ERROR_PenPlot_InvalidEye	
303	VP_ERROR_PenPlot_OutOfRange	
304	VP_ERROR_PenPlot_Unknown_IDM	
305	VP_ERROR_PenPlot_InitInstance_NULLpRect	



Arrington Research

	500s are License & Decrypt errors	
502	VP_ERROR_CryptFileEndError	
503	VP_ERROR_CryptFileDataInvalid	May have unreadable characters in file
504	VP_ERROR_LicenseDataOpenError	
505	VP_ERROR_DecryptCreateKeyContainer	
506	VP_ERROR_DecryptServiceHandle	
507	VP_ERROR_DecryptHeaderLength	
508	VP_ERROR_DecryptMemoryAlloc	
509	VP_ERROR_DecryptFileHeader	
510	VP_ERROR_DecryptCryptImportKey	
511	VP_ERROR_CryptCreateHash	
512	VP_ERROR_CryptHashData	
513	VP_ERROR_CryptDeriveKey	Windows98 error, must use Windows XP
514	VP_ERROR_CryptDestroyHash	
515	VP_ERROR_DecryptOutOfMemory	
516	VP_ERROR_DecryptReadingCiphertext	
517	VP_ERROR_CryptDecrypt	
518	VP_ERROR_DecryptSourceClose	
519	VP_ERROR_CryptReleaseContext	
520	VP_ERROR_LicenseDirectoryMissing	
521	VP_ERROR_LicenseFileMissing	
522	VP_ERROR_LicenseFileInvalid	
523	VP_ERROR_LicenseNumberInvalid	
524	VP_ERROR_CryptDestroyKey	Was previously 518
525	VP_ERROR_NoCameraFound	
526	VP_ERROR_CameralInitFailed	
	600s are EyeMovie	
601	VP_ERROR_MovieFileNotFound	
602	VP_ERROR_MovieFileReadError	
603	VP_ERROR_MovieFileBadFormat	
604	VP_ERROR_MovieFileMemoryWarning	
605	VP_ERROR_MovieFileOpenError	diagnosis of exclusion
606	VP_ERROR_EyeMovieSetThreadPriority	
607	VP_ERROR_EyeMovie_NoVideoStream	
608	VP_ERROR_EyeMovie_PostQuitThread	
609	VP_ERROR_EyeMovie_PostPauseThread	
	700s video errors	
701	VP_ERROR_VideoInitializationFailed	was previously 101
702	VP_ERROR_VideoStartupFailed	was previously 102
703	VP_ERROR_VideoChannelFailed	was previously 103
	900s ethernet errors	
901	VP_ERROR_EthernetError	Currently not used
	1000s are for bad function arguments	
1001	VP_ERROR_BadCalibrationSpeedValue	Must be in range: (1-400)
1002	VP_ERROR_BadCalibrationISIValue	Must be in range: (0-9)
1003	VP_ERROR_BadCalibrationWarningTimeValue	Must be in range: (1-100)
	2000s are for Date & Time	
2001	VP_ERROR_StringFormat_LongDate	
2003	VP_ERROR_StringFormat_TimeOfDay	
2004	VP_ERROR_HistoryTimeStamp	
2005	VP_ERROR_HistoryTimeStamp	
2006	VP_ERROR_GetDateFormat	
2007	VP_ERROR_GetTimeFormat	



Arrington Research

2008	VP_ERROR_GetSystemTimes	
	3000s utilities	
3001	VP_ERROR_EnsureSubDirectory	
3002	VP_ERROR_EnsureSubDirectoryFolder	
3003	VP_ERROR_SettingsFolderCreatedNotice	
	4000s LC1 errors	
4001	VP_ERROR_LC1_InvalidVideoChannel	
4002	VP_ERROR_LC1_AllocImageBufferError	
4003	VP_ERROR_LC1_AddBufferToSequenceError	
4004	VP_ERROR_LC1_FreeBufferInSequenceError	
4010	VP_ERROR_SegmentGlint_Buffer	
4101	VP_ERROR_uEye_ImageMemoryNotSequential	
4151	VP_ERROR_uEye_Unsuitable_CameralID	
4152	VP_ERROR_uEye_Duplicate_CameralID	
4153	VP_ERROR_uEye_OutOfRange_CameralID	
5001	VP_ERROR_ROI_IndexErr	
6001	VP_ERROR_SensorayNeedsRestarting	
6002	VP_ERROR_SensorayDriverNeedsUpdate	
7001	VP_ERROR_StateEngine_PostQuitThread	
7002	VP_ERROR_StateEngine_PostPauseThread	
7003	VP_ERROR_StateEngine_PostResumeThread	
	9300s LC1 Capture Board errors	
9301	VP_ERROR_LC1_Not32BitColor	
9302	VP_ERROR_LC1_ExitBoard	
9303	VP_ERROR_LC1_SetThreadPriority	
9304	VP_ERROR_LC1_NoVideoChannels	
	9400s ViewPointMovie (VPM) errors	
9401	VP_ERROR_VPM_WriteFrame	
9402	VP_ERROR_VPM_OpenExisting	
9403	VP_ERROR_VPM_ReadFrame	
	9500 AVI 2.0 errors	
9500	VP_ERROR_AVI2_DxDIBService_CoCreate	
9501	VP_ERROR_AVI2_DxDIBService_Provider	
9502	VP_ERROR_AVI2_Writer_BadVideoChan	
9503	VP_ERROR_AVI2_Writer_ServiceFailed	
9504	VP_ERROR_AVI2_Writer_CodecListIndex	
9505	VP_ERROR_AVI2_Writer_CodecSetFailed	
9506	VP_ERROR_AVI2_Writer_RateSetFailed	
9507	VP_ERROR_AVI2_WriterCreateFileFail	
9508	VP_ERROR_AVI2_WriterFailedToAppend	
9509	VP_ERROR_AVI2_GetFrameByIndexFailed	
9510	VP_ERROR_AVI2_OkayToAccessQ_VidChan	
9511	VP_ERROR_AVI2_MissingLibsFolder	
9512	VP_ERROR_AVI2_FailedToGetCodecs	
9513	VP_ERROR_AVI2_FailedToSetupMenuCodecs	
9514	VP_ERROR_AVI2_TooManySystemCodecs	



Arrington Research

	9600s Generic Movies	
9601	VP_ERROR_MovieTypeObsolete_AVI1	
9602	VP_ERROR_MovieFileSizeZero	
	9700s COM, DCOM etc.	
9701	VP_ERROR_COM_ThreadModeChanged	
9701	VP_ERROR_COM_GetProcAddress_DllRegisterServer	
9701	VP_ERROR_COM_DllRegisterServer	
	9800s Parser, SettingsFile, CLI	
9801	VP_ERROR_CLI_ObsoleteCommand	
9802	VP_ERROR_CLI_UnmatchedQuotesOnLine	
9803	VP_ERROR_CLI_UnmatchedBracesOnLine	
	9900s DLL Errors	
9900	VP_ERROR_DLL_InsertMessageError	
9901	VP_ERROR_DLL_SetRealtimeBufferSizeError	
9902	VP_ERROR_DLL_InsertCallbackError	

Chapter 24. Hardware Installation

This chapter describes the procedure for the video capture hardware and driver installation process and *ViewPoint EyeTracker* ® software installation.

IMPORTANT: The display must be set to True Color (32 bit).

IMPORTANT: Make sure that the latest .NET version is installed.

24.1 USB-220 & USB-400 Installation

There are four main steps to this installation.



1. Connect the camera to your computer

Connect the USB cable to your computer and to the camera. No separate power is required; the camera takes power from the computer via the USB cable.

2. Install the USB camera drivers

When the Windows Found New Hardware wizard appears, cancel. Browse to the location on your computer where you have copied the contents of the *ViewPoint* disk.

Find the .../ViewPoint/Drivers/ folder and install the **uEye** driver. Follow the onscreen instructions to install the drivers.

3. Ensure the CameraID is set properly for each camera.

With Binocular and SceneCamera systems (as of *ViewPoint* version 2.9.3.115), each camera must be set with a unique **CameraID** number that specifies its use (1=EyeA, 2=EyeB, or 3=Scene). Normally this is done before the product is shipped, but can be done by the user with the **IDS Camera Manager** program.

In *ViewPoint*, to view each camera **SerialNumber** and CameraID, select menu item: *Help > Info > SysInfo tab* and look at the *Hardware:* line.

4. Power the illuminator

Connect the USB cable to your computer and to the illuminator. No separate power is required; the camera takes power from the computer via the USB cable.

5. Run ViewPoint

Run *ViewPoint* by double clicking the **ViewPoint-220_64.exe**. The *EyeCamera* window will show a video image.



The *EyeCamera* window Eyelimage is always displayed at 320 X 240, regardless of the mode selected. The available modes of operation for this product depend upon the license that was purchased, but in general may include:

Table 17. USB-220 / USB-400 Spatial and Temporal Resolutions

<i>Setting</i>	<i>Temporal Resolution (Hz)</i>	<i>Internal Processing Spatial Resolution</i>
90	90	320 x 240
220	220	320 x 200
350	350	240 x 150
400	400	200 x 120

Note that you may easily upgrade to a higher speed system at any time, by simply purchasing a new license that enables higher speeds on your existing camera. Please enquire for details and pricing.

24.2 60 Hz Video Capture

The *ViewPoint EyeTracker*® 60 Hz products require the special high performance video capture device supplied by *Arrington Research*. It will not work with other video capture devices.

The 60 Hz models provide three modes of operation that provide a selection of resolutions and sampling rates. Which mode you choose will depend on your research or project requirements. You may achieve better performance if you calibrate using the **High Precision Mode**.

- ④ **Setup Mode:** **Interlaced**
Temporal Resolution: 30Hz
Internal Processing: 320 x 240
- ④ **High Precision Mode:** **Interlaced**
Temporal Resolution: 30Hz
Internal Processing: 640 x 480
- ④ **High Speed Normal Mode:** **De-interlaced**
Temporal Resolution: 60Hz
Internal Processing: 320 x 240

The monitor icon > Mode at the bottom of the *EyeCamera* window tool bar can be used to select the operating mode. Historically, when computers were slower, the image of the eye only appeared in Setup Mode; the other modes only displayed the overlay graphics, to save CPU time. This is no longer the case and the term is somewhat misleading because “setup” can be done in any of the modes. This mode may better be described as a “low CPU” mode; nevertheless, the old terminology is retained for the present.

24.2.1 Camera Systems

ViewPoint EyeTracker® systems that include an analog NTSC video camera and video capture device can be used in any country, including those that have PAL or other video standard. There is no problem because the NTSC video camera and the NTSC video capture device makeup a closed video system.

24.2.2 Using with Third Party Video Cameras & Signals

The *ViewPoint PC-60* analog video capture hardware allows video input that includes the PAL and SECAM standards, as well as the default NTSC standard. The *ViewPoint USB-60x3 SilverBox* currently only supports NTSC.

To change the video input type for the *EyeCamera*, the user should use the monitor icon on the *EyeCamera window* > *Monitor Icon button* > *Video Standard* > * to select the standard that corresponds to the type of video camera, video recorder, etc., that is used. The following CLI command may also be used:

```
scene_videoStandard <format> // where <format> is one of: NTSC, PAL, SECAM
```

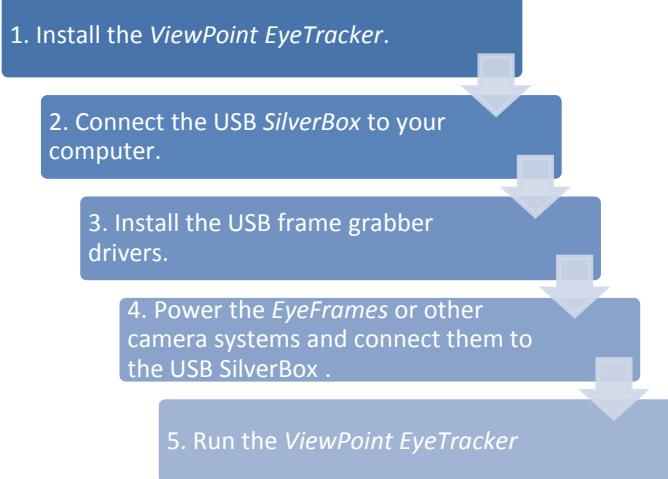
The selected video standard is stored in the preferences file and will be used as the default when *ViewPoint* is next run.

The default setting is NTSC and this should not be modified unless third party video equipment is used that specifies a different video standard.

Specific installation instructions for the 60 Hz products *USB-60x3*, and *PC-60* are found in the following sections.

24.3 USB-60x3 (SilverBox) Installation and Set Up

There are five main steps to the USB-60x3 installation.



- 1. Install *ViewPoint EyeTracker*®:** Copy the entire *ViewPoint* folder to your computer.
- 2. Connect the USB *SilverBox* frame grabber to your computer:** Connect the USB cable to your computer and to the silver box. No separate power is required; the frame grabber takes power from the computer via the USB cable.
- 3. Install the drivers for the USB frame grabber:** When the *Windows Found New Hardware* wizard appears, cancel. Browse to the location on your computer where you have copied the contents of the *ViewPoint* disk.
Run *Setup.exe* in folder: .../ViewPoint/Drivers/ Camera Drivers - USB-60x3 SilverBox/
Follow the onscreen instructions to install the drivers.
- 4. Power and connect the EyeFrames:** Connect the EyeFrame cables to the USB framegrabber as below and then to the 12V transformer.
- 5. Run *ViewPoint*:** Run *ViewPoint* by double clicking the *ViewPoint-USB-60x3.exe* -- the *EyeCamera* window will show a video image..

24.4 PC-60 (PCI card) Installation and Set Up

Important Install the Drivers, before installing the card

1. If you are updating from version 1.x of *ViewPoint EyeTracker*® or have had another BT848 video capture device previously installed, you must FIRST remove all of the video capture software and drivers from your computer. SECOND, after the driver software has been removed, physically remove the old video capture device from your computer before proceeding with the new frame grabber installation.
2. Disconnect from the network prior to installing the drivers.

A. Installing the New Driver

1. Insert the *ViewPoint EyeTracker*® CD-ROM into your CD-ROM drive.
2. Browse CD to the top level Drivers folder.
3. Run Setup for your Operating System
 - a. Windows XP F:\Drivers\PC-60\Pre vista drivers\lc1_2\English\Disk1\Setup.exe
 - b. Win 7 32bit F:\Drivers\PC-60\Windows 7 32 BIT\Setup\Setup.exe
 - c. Win 7 64 bit F:\Drivers\PC-60\Windows 7 64 BIT\Setup\Setup.exe
(If F is not the device letter of your CD-ROM drive, substitute with the correct drive letter).
4. Follow the onscreen instructions to complete installation.
5. Restart your computer when instructed to.

B. Installing ViewPoint EyeTracker® Software

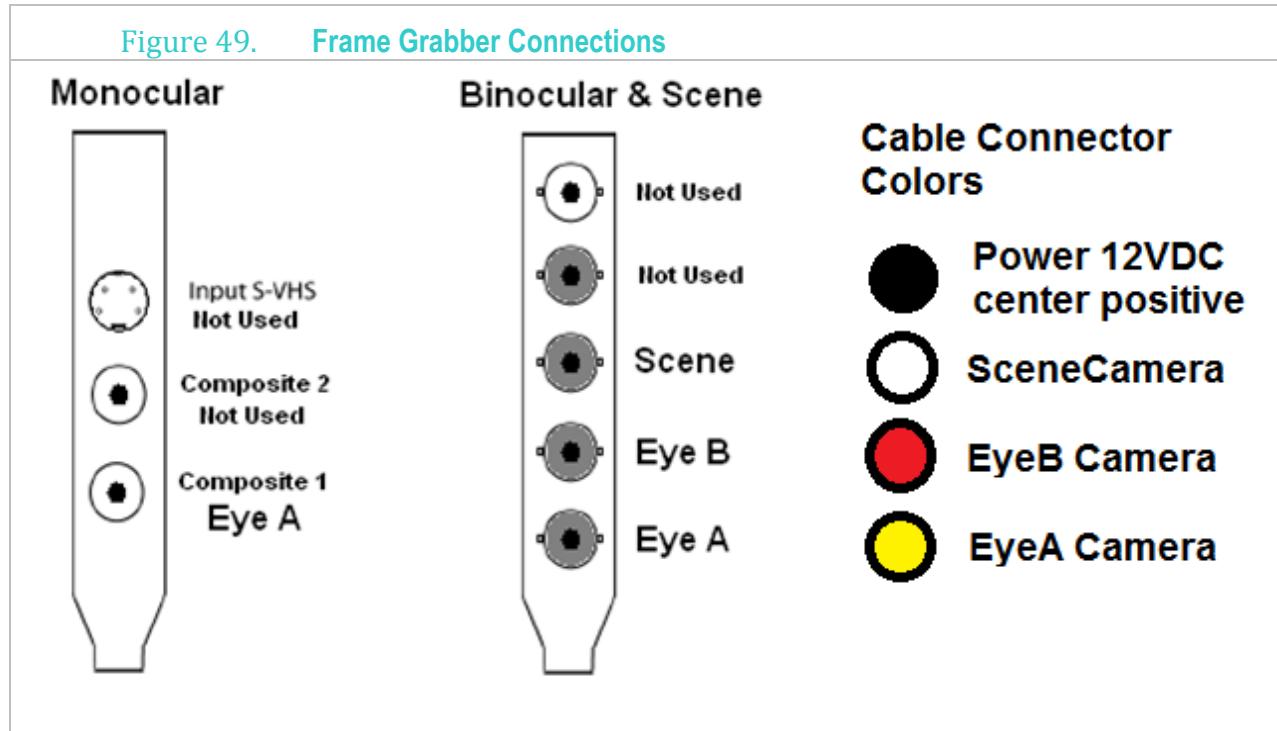
1. Copy the *ViewPoint EyeTracker*® folder from the CDROM to the hard drive of your computer.
2. You may start the program immediately by double clicking the icon of the *ViewPoint* application program.

This directory structure must be maintained for proper functioning of the software. The *ViewPoint* software will not run without the .dll file. Do not make illegal copies.

C. Installing the New Frame Grabber

1. Turn off the computer, and then disconnect the power cable.
2. Remove the cover panel from your computer. If necessary, consult your computer system manual for instructions.
3. Remember to discharge your body's static electricity by touching the metal area of the computer chassis.
4. Select an empty PCI or PCIe slot card depending on the card and remove the slot cover.
5. Install the card into the slot, paying particular attention that the card is inserted and seated correctly.
6. Screw the card into place.
7. Replace the cover panel.
8. Reconnect the power cable and turn on the computer.
9. If the update device driver wizard starts. Click cancel.
10. If the Windows "Found New Hardware Wizard" asks you if you would like to connect to Windows update to search for the drivers select "No, not at this time" and "Next".
11. Select "Install the software automatically" and "Next".
12. At the next dialogue box, with the top line item highlighted select "Next".
13. At the "not digitally signed" warning select "Continue Anyway".
14. Select Finish.

NOTE: you will have to repeat the above steps for each input if you have a binocular or scene camera version of the eye tracker.



D. Troubleshooting

- Problems with *ViewPoint EyeTracker*.

Device Manager > **Sound, video and game controllers.** Manually install drivers

All drivers must be installed for the card, even if they will not be used. With a monocular card, two video and two Audio drivers should appear; with a Binocular & Scene cards, four Audio and four Video drivers should be present.

Figure 50. All drivers must be installed, even if they are not being used.



Chapter 25. Latency

The *ViewPoint~Voltage* program includes a test to measure latency. This software alternately switches two LEDs On and Off and then measures the time that it takes to receive an ROI event from *ViewPoint* (in bright pupil mode), which indicates that the light position changed. This “round trip” latency time is quantitative and reproducible.

Differences in latency come from different hardware systems and from the exposure time that is set on digital cameras – the faster the camera speed the lower the latency. Here is a table that provides the results of some of our tests. You may want to reproduce these tests on the system that you are using.

Table 18. Latency of various systems

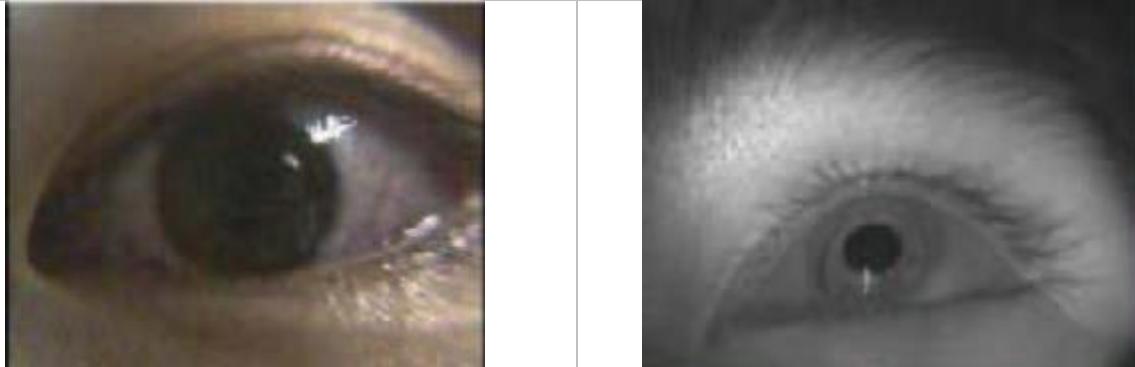
Product	Average Latency (milliseconds)
USB-400	6
USB-220	10
PC-60 (same for 30 or 60 Hz)	55 (varies 50 to 60)
USB-60x3 @ 60 Hz	65 (varies 50 to 80)

Chapter 26. Safety

26.1 Infrared Light

The value of using infrared light is illustrated in [Figure 51](#). The left side of the figure shows an image in normal light; for this subject the pupil of the eye is almost impossible to discriminate from the dark iris. The right side of the figure shows an image of the same eye, but viewed with an infrared sensitive camera under infrared lighting conditions; the pupil is easily discriminated. Note that in each case the subject is wearing a contact lens.

Figure 51. Infrared light allows for pupil discrimination



There should always be the utmost concern for the safety of the subject. The issue of safe limits of infrared (IR) irradiance is frequently discussed.

10 mW / cm² is probably the safe maximum figure for corneal exposure over a prolonged period (Clarkson, T.G. 1989, Safety aspects in the use of infrared detection systems, I. J. Electronics, 66, 6, 929-934).

The infrared corneal dose rate experienced out of doors in daylight is of the order of 10⁻³ W / cm⁻². Safe chronic ocular exposure values particularly to the IR-A (750-1400nm), probably are of the order of 10⁻² W / cm⁻² (D.H. Sliney & B.C. Freasier, Applied Optics, 12:1, 1973).

ISO/DIS 10342 (page 7) gives maximum recommended fundus irradiance for use in Ophthalmic Instruments of 120 mW / sq cm but this is for short-term exposure.

All IR-illuminator and camera systems provided by Arrington Research, Inc. are designed to be well within safe limits of exposure.

See also: COGAIN IST-2003-511598

Chapter 27. ARI Software License

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT "AGREEMENT" CAREFULLY BEFORE USING THE SOFTWARE. BY USING ALL OR ANY PART OF THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU ACQUIRED THE SOFTWARE ON TANGIBLE MEDIA (e.g. CD) WITHOUT AN OPPORTUNITY TO REVIEW THIS AGREEMENT AND YOU DO NOT ACCEPT THIS AGREEMENT, YOU MAY OBTAIN A REFUND OF ANY AMOUNT YOU ORIGINALLY PAID IF YOU: (A) DO NOT USE THE SOFTWARE AND (B) RETURN IT, WITH PROOF OF PAYMENT, TO THE LOCATION FROM WHICH IT WAS OBTAINED WITHIN THIRTY (30) DAYS OF THE PURCHASE DATE.

1. Definitions: "Software" means (a) all of the contents of the files, disk(s), CD-ROM(s) or other media with which this Agreement is provided, including but not limited to (i) ARI or third party computer information or software; (ii) digital images, stock photographs, clip art, sounds or other artistic works ("Stock Files"); and (iii) related explanatory written materials or files ("Documentation"); and (b) upgrades, modified versions, updates, additions, and copies of the Software, if any, licensed to you by ARI (collectively, "Updates"). "Use" or "Using" means to access, install, download, copy or otherwise benefit from using the functionality of the Software in accordance with the Documentation. "Permitted Number" means one (1) unless otherwise indicated under a valid license (e.g. volume license) granted by ARI. "Computer" means an electronic device that accepts information in digital or similar form and manipulates it for a specific result based on a sequence of instructions. "ARI" means Arrington Research, Inc., an Arizona corporation.

2. Software License As long as you comply with the terms of this Agreement, ARI grants to you a non-exclusive license to use the Software for the purposes described in the Documentation. You may install and use a copy of the Software on your compatible computer, up to the Permitted Number of computers; You may make one backup copy of the Software, provided your backup copy is not installed or used on any computer. The software accompanying this Agreement, whether on disk, on compact disk, in read only memory, or any other media, the related documentation and other materials (collectively, the "ARI Software") are licensed, not sold, to you by ARI. The ARI Software in this package and any copies, modifications and distributions which this Agreement authorizes you to make are subject to this Agreement.

3. Intellectual Property Rights. The Software and any copies that you are authorized by ARI to make are the intellectual property of and are owned by ARI. The structure, organization and code of the Software are the valuable trade secrets and confidential information of ARI. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. You may not copy the Software, except as set forth in Section 2 ("Software License"). Any copies that you are permitted to make pursuant to this Agreement must contain the same copyright and other proprietary notices that appear on or in the Software. You also agree not to reverse engineer, decompile, disassemble or otherwise attempt to discover the source code of the Software. Trademarks can only be used to identify printed output produced by the Software and such use of any trademark does not give you any rights of ownership in that trademark. Except as expressly stated above, this Agreement does not grant you any intellectual property rights in the Software. The ARI software may only be used by you for the purpose described herein and may not be disclosed to any third party or used to create any software which is substantially similar to the expression of the Software.

4. Transfer. You may not rent, lease, sublicense or authorize all or any portion of the Software to be copied onto another user's computer except as may be expressly permitted herein. You may, however, transfer all your rights to use the Software to another person or legal entity provided that: (a) you also transfer each this Agreement, the Software and all other software or hardware bundled or pre-installed with the Software, including all copies, Updates and prior versions, to such person or entity; (b) you retain no copies, including backups and copies stored on a computer; and (c) the receiving party accepts the terms and conditions of this Agreement and any other terms and conditions upon which you legally purchased a license to the Software; (d) you obtain prior written permission from ARI. Notwithstanding the foregoing, you may not transfer education, pre-release, or not for resale copies of the Software.

5. Limited Warranty on Media: ARI warrants to the person or entity that first purchases a license for the Software for use pursuant to the terms of this agreement, that the software is recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of original purchase. Non-substantial variations of performance from the Documentation does not establish a warranty right. THIS LIMITED WARRANTY DOES NOT APPLY TO UPDATES, OR NOT FOR RESALE (NFR) COPIES OF SOFTWARE. To make a warranty claim, you must request a return merchandize authorization number, and return the Software to the location where you obtained it along with proof of purchase within such ninety (90) day period. The entire liability of ARI and your exclusive remedy shall be limited to either, at ARI's option, the replacement of the Software or the refund of the license fee you paid for the Software. THE LIMITED WARRANTY SET FORTH IN THIS SECTION GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE ADDITIONAL RIGHTS WHICH VARY FROM JURISDICTION TO JURISDICTION.



6. Disclaimer of Warranty. Some of the ARI Software may be designed as alpha, beta, development, continuing development, pre-release, untested, not fully tested or research only versions of the ARI Software. Such ARI Software may contain errors that could cause failure of loss of data, and may be incomplete or contain inaccuracies. You expressly acknowledge and agree that use of the ARI Software is at your sole risk. The ARI Software is provided "AS IS" and without warranty of any kind and ARI and ARI's licensor(s) EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. ARI DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE ARI SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE ARI SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE ARI SOFTWARE WILL BE CORRECTED. FURTHERMORE, ARI DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE ARI SOFTWARE OR IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY ARI OR AN ARI AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANYWAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE ARI SOFTWARE PROVE DEFECTIVE, YOU (AND NOT ARI OR AN ARI AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. THE LICENSE FEES FOR THE ARI SOFTWARE REFLECT THIS ALLOCATION OF RISK. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

7. Limitation of Liability. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL ARI BE LIABLE FOR ANY INCIDENT, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE ARI SOFTWARE, EVEN IF ARI OR AN ARI AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW LIMITATIONS OR EXCLUSION OF LIMITED LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. In no event shall ARI's total liability to you for all damages, losses and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the license fee that you paid for the Software.

8. High Risk Activities: Effort has been made to provide a bug-free product. Nevertheless, this software is not fault tolerant and is not designed, manufactured or intended for use or resale in the operation of nuclear facilities, aircraft navigation or communications systems, or air traffic control, or medical treatment and diagnosis, or for any other use where the failure of the Software could lead to death, personal injury, damage to property or environmental damage ("High Risk Activities"). ARI specifically disclaim any express or implied warranty of fitness for High Risk Activities.

© Arrington Research, Inc. All rights reserved 100 **9. Export Law Assurances.** You agree that the ARI Software will not be exported outside the United States except as authorized by United States law. You also agree that ARI Software

that has been rightfully obtained outside the United States shall not be re-exported except as authorized by the laws of the United States and of the jurisdiction in which the ARI Software was obtained.

10. Controlling Law and Severability. This Agreement shall be governed by the laws of the United States. If for any reason a court of competent jurisdiction finds any provision, or portion thereof, to be unenforceable, the remainder of this Agreement shall continue in full force and effect.

11. Complete Agreement. This Agreement constitutes the entire agreement between the parties with respect to the use of the ARI Software and supersedes all prior or contemporaneous understandings regarding such subject matter. No amendment to or modification of this Agreement will be binding unless in writing and signed by ARI.

Chapter 28. Third Party Licenses

From time to time, some portions of the *ViewPoint EyeTracker* ® code may utilize third party libraries, or modifications thereof, that have their own License Agreements. These are included here below.

Intel License Agreement For Open Source Computer Vision Library

Copyright (C) 2000-2005, Intel Corporation, all rights reserved.

Third party copyrights are property of their respective owners.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistribution's of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistribution's in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The name of Intel Corporation may not be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the Intel Corporation or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

AT&T License Agreement for 2D Convex Hull code

Some 2D convex hull code is modified from what was written by Ken Clarkson. Copyright (c) 1996 by AT&T.

Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software.

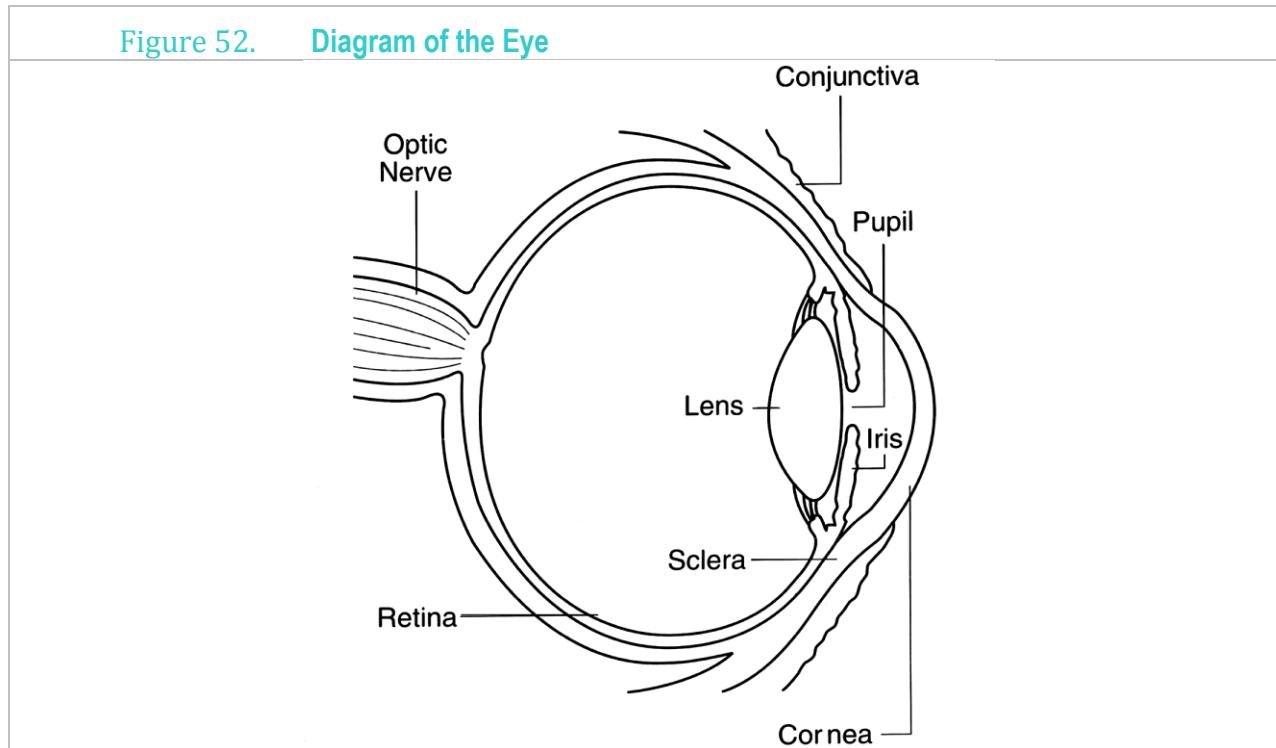
* THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

Chapter 29. Appendix

29.1 Diagram of the Eye

We have included a diagram below showing the anatomy of the eye to help with understanding some of the terms used in this *UserGuide*. See [Figure 52](#).

This diagram has been included with thanks to the National Eye Institute and the National Institutes of Health (NEI). The NEI website includes a valuable resource of photographs and images.



29.2 Mapping to GazeSpace

It is often necessary to determine where a person is looking, that is, to determine their *ViewPoint*, also called the *Point of Regard* (POR), the *Position of Gaze* (POG), *GazePoint*, *Fixation Location*, etc. This task is performed by using a mathematical function to map the eye position signal in the *EyeSpace* coordinates of the video image to the gaze point in the *GazeSpace* coordinates of the visual stimulus. There are many algorithms that can be used to perform such a mapping and many of them are company proprietary. By far, the best algorithms are non-linear. This is because the eye movements are rotational, i.e., the translation of the eye position signal that is apparent to the camera is a trigonometric function of the subject's gaze angle. Moreover, the camera angle may provide an oblique line-of-sight. The *ViewPoint EyeTracker* ® employs one of the most powerful and robust methods available. See [Table 19: Schematic of the ViewPoint EyeTracker® System \(Head Fixed / HMD\)](#).

29.3 Schematic Bloc Description of *ViewPoint*

[Table 19](#) shows how the *ViewPoint EyeTracker* ® works in a typical head fixed or HMD configuration. The numbers in this section refer to the item or block numbers in the figure.

The infrared light source (*item 1*) serves to both illuminate the eye (*item 2*) and also to provide a specular reflection from the surface of the eye, i.e., from the smooth cornea. In Dark Pupil Mode, the pupil acts as an infrared sink that appears as a black hole; see [Figure 51](#). In Bright Pupil Mode, the “red eye” effect causes the pupil to appear brighter than the iris (note that a different camera and illuminator configuration is required for bright pupil operation).

The video signal from the camera (*item 3*) is digitized by the video capture device (*item 4*) into a form that can be understood by a computer. The computer takes the digitized image and applies image segmentation algorithms (*item 5*) to locate the areas of pupil and the bright corneal reflection (glint). Additional image processing (*item 6*) locates the centers of these areas and also calculates the difference vector between the center locations. A mapping function (*item 7*) transforms the eye position signals (*item 6*) in *EyeSpace* coordinates to the subject’s *GazeSpace* coordinates (*item 8*).

Eye movements are classified (*item 13*): fixation, drift, or saccade and blinks are detected.

Next, the program tests to determine whether the Gaze Point is inside of any of the Region of Interest (ROI) that the user has defined.

The calibration system (*item 12*) can be used to present calibration stimuli via (*item 10*) to the user and to measure the eye position signals (*item 6*) for each of the StimulusPoints. These data are then used by (*item 12*) to compute an optimal mapping function for mapping to Position of Gaze in *GazeSpace* (*item 7*).

Table 19. Schematic of the *ViewPoint EyeTracker®* System (Head Fixed / HMD)

