ViewPoint E-Prime™

a ***ViewPoint EyeTracker*** ® interface

to the *E-Prime* ® application

Software User Guide

Arrington Research

ArringtonResearch

ViewPoint E-Prime manual	2.8.3	27-Oct-05
Tested with ViewPoint version	2.8.3.31	27-Oct-05

Arrington Research, Inc.
27237 N 71st Place, Scottsdale, AZ 85262
United States of America
www.ArringtonResearch.com
Phone 480.985.5810
info@ArringtonResearch.com
support@ArringtonResearch.com

.

1. Getting Started

1.1. About this Manual

The documentation describes a DLL based interface between ***ViewPoint EyeTracker®*** (*ViewPoint*) and ***E-Prime®*** .

It is assumed that the user is familiar with *ViewPoint,* and *E-Prime.* The user should consult the respective manuals for information about how to use these products.

1.2. ViewPoint E-Prime Interface

It is assumed that the user already has *ViewPoint* and *E-prime* installed. The *ViewPoint E-Prime interface* provides a rich set of functions that can be used. The list of functions available is mentioned in the next section. The functions have to be declared in E-Prime before they are used. A sample code is given below for declaring and using a function:

```
'------------------------------------------------------

' Declaration of functions and variables

'------------------------------------------------------

 Type VPX_RealPoint

  x as Single

  y as Single

 End Type

Declare Function vpxs_GetGazePoint2 Lib
   "VPX_InterApp.dll" Alias "vpxs_GetGazePoint2" (ByVal
   eyetype as Integer,ByRef pa as VPX_RealPoint) As
   Integer

Declare Function vpxs_SendCommandString Lib
   "VPX_InterApp.dll" Alias "vpxs_SendCommandString"
   (ByVal lpcmd as String) As Integer

'--------------------------------------------------------
```

The functions can then be used in any InLine object (the InLine object is used to insert user-written E-Basic script into an E-Prime experiment). Refer to the Example.es provided.

2. *ViewPoint E-Prime Interface* functions

2.1. Functions List and Online Help

The *ViewPoint E-Prime Interface* provides a rich set of functions that can be used

The next section describes each function in detail.

2.2. Individual Function Descriptions

2.2.1. SendCommandString

```
Dim ok as Integer
ok = vpxs_SendCommandString( str )

eg: vpxs_SendCommandString('autothreshold');
```

This provides a mechanism to send command strings to the *ViewPoint* command line parser. An external program can completely control the *ViewPoint Eyetracker* via the command line interface – every menu item, button, etc. in *ViewPoint* has an equivalent command line interface. See the *ViewPoint EyeTracker Manual* for a list of commands.

2.2.2. GetGazePoint

```
Dim ok as Integer
ok = vpxs_GetGazePoint(VPX_RealPoint* gazePoint)
ok = vpxs_GetGazePoint2( VPX_EyeType eye, VPX_RealPoint* gazePoint)
ok = vpxs_GetGazePointSmoothed( VPX_RealPoint* gazePoint  )
ok = vpxs_GetGazePointSmoothed2( VPX_EyeType eye, VPX_RealPoint* gazePoint )
```

Retrieves the calculated position of gaze, for either Eye_A (eyetype=0) or Eye_B (eyetype=1) if binocular mode is on. Monocular ViewPoint uses Eye_A by default.

See also:

```
vpxs_GetGazeAngle
vpxs_GetPupilPoint
vpxs_GetGlintPoint
```

2.2.3. GetGazeAngle

```
Dim ok as Integer
ok = vpxs_GetGazeAngle( VPX_RealPoint* gp)
ok = vpxs_GetGazeAngle2( VPX_EyeType eye, VPX_RealPoint* gp )
ok = vpxs_GetGazeAngleSmoothed ( VPX_RealPoint* gp)
ok = vpxs_GetGazeAngleSmoothed2( VPX_EyeType eye, VPX_RealPoint* gp )
```

Retrieves the calculated angle of gaze, for either Eye_A, or Eye_B if binocular mode is on. Monocular ViewPoint uses Eye_A by default.
Note: The angles are from trigonometric calculations based on the values that the user has measured and set for the window size (horizontal & vertical) and the viewing distance.

2.2.4. GetFixationSeconds

```
Dim Fixationsec as Integer
Fixationsec = vpxs_GetFixationSeconds2( VPX_EyeType eye, double* seconds )
```

Retrieves the number of seconds that the total velocity has been below the VelocityCriterion for either Eye_A, or Eye_B if binocular mode is on. Monocular ViewPoint uses Eye_A by default. .
A zero value indicates a saccade is occurring.

See also:
`vpxs_GetTotalVelocity`

2.2.5. GetTotalVelocity

```
Dim Velocity as Integer
Velocity = vpxs_GetTotalVelocity(double* velocity)
Velocity = vpxs_GetTotalVelocity2( VPX_EyeType eye, double* velocity )
```

Retrieves the total velocity of movement in the (x,y) plane. That is, the first derivative of the (smoothed) position of gaze for either Eye_A, or Eye_B if binocular mode is on. Monocular ViewPoint uses Eye_A by default.

See also:
`vpxs_GetComponentVelocity`
`vpxs_GetFixationSeconds2`

2.2.6. GetComponentVelocity

```
Dim ok as Integer
ok = vpxs_GetComponentVelocity(VPX_RealPoint *velocityComponents)
ok = vpxs_GetComponentVelocity2(VPX_EyeType eye,VPX_RealPoint *velocityComponents)
```

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.

See also:

`vpxs_ GetTotalVelocity`

2.2.7. GetPupilSize

```
Dim ok as Integer
ok = vpxs_GetPupilSize( VPX_RealPoint *dims )
ok = vpxs_GetPupilSize2( VPX_EyeType eye, VPX_RealPoint *dims )
```

Retrieves the raw size of the oval fit to the pupil for either Eye_A, or Eye_B if binocular mode is on. Monocular ViewPoint uses Eye_A by default.
The x- y-size values are normalized with respect to the EyeSpace dimensions that have a 4:3 aspect, so the x- and y-values are anisotropic. To obtain the aspect ratio of the pupil, rescale: (aspect = ps.x / (ps.y * 0.75)

See also:

`vpxs_GetPupilAspectRatio`
`vpxs_GetPupilAspectRatio2`

2.2.8. GetPupilAspectRatio

```
Dim aspect as Integer
aspect = vpxs_GetPupilAspectRatio( double *ar )
aspect = vpxs_GetPupilAspectRatio2( VPX_EyeType eye,double *ar )
```

Retrieves the dimensionless value of pupil aspect ratio. The ratio is independent of the EyeCamera window shape. Note: a circular pupil will produce a value of 1.0

See also:

`vpxs_GetPupilSize`
`vpxs_GetPupilSize2`

2.2.9. GetPupilPoint

```
Dim ok as Integer
ok = vpxs_GetPupilPoint( VPX_RealPoint *rawPupilLoc )
ok = vpxs_GetPupilPoint2( VPX_EyeType eye, VPX_RealPoint *rawPupilLoc )
```

Retrieves the **raw** normalized (x,y) location of the center of the pupil (center of the oval fit to the pupil) in the EyeSpace, for either Eye_A, or Eye_B if binocular mode is on. Monocular ViewPoint uses Eye_A by default.

2.2.10. GetGlintPoint

```
Dim ok as Integer
ok = vpxs_GetGlintPoint( VPX_RealPoint *rawGlintLoc )
ok = vpxs_GetGlintPoint2( VPX_EyeType eye, VPX_RealPoint *rawGlintLoc )
```

Retrieves the **raw** normalized (x,y) location of the center of the glint (center of the oval fit to the glint) in the EyeSpace, for either Eye_A or Eye_B if binocular mode is on. Monocular ViewPoint uses Eye_A by default.

2.2.11. GetDiffVector

```
Dim ok as Integer
ok = vpxs_GetDiffVector( VPX_RealPoint *dv )
ok = vpxs_GetDiffVector2( VPX_EyeType eye, VPX_RealPoint *dv )
```

Retrieves the **raw** normalized vector difference between the centers of the pupil and the glint in the EyeSpace, for either Eye_A or Eye_B if binocular mode is on.
Monocular ViewPoint uses Eye_A by default.

2.2.12. GetTorsion

```
vpxs_GetTorsion ( double *degrees )
vpxs_GetTorsion2( VPX_EyeType eye, double *degrees )
```

Retrieves torsion in degrees for either Eye_A or Eye_B if binocular mode is on.
Monocular ViewPoint uses Eye_A by default. The ViewPoint torsion measurement must be turned on, it is off by default to save processing time.

2.2.13. GetDataQuality

```
Dim q as Integer
q = vpxs_GetDataQuality2( VPX_EyeType eye, int *quality )
```

Retrieves the quality code for the eye data. See `VPX.h` for a list of data quality constants.

2.2.14. GetDataTime

```
vpxs_GetDataTime2( VPX_EyeType eye, double *dataTime)
```

Retrieves the time (double floating point, high precision time, in seconds) that the video frame became available for the current data point, before video image processing and other calculations were done.

See also:
`vpxs_GetDataDeltaTime2`

2.2.15. GetDataDeltaTime

```
vpxs_GetDataDeltaTime2( VPX_EyeType eye, double* deltaTime )
```

Retrieves the time interval (double floating point, high precision time interval, in seconds) between the last two *dataTime* values.

2.2.16. GetPupilCentroid

```
vpxs_GetPupilCentroid2( VPX_EyeType eye, VPX_RealPoint *centroid )
```

Provides **raw** data access to the normalized centroid of the pupil threshold scan in the EyeSpace window, regardless of what subsequent processing options are selected.

2.2.17. GetGlintCentroid

```
vpxs_GetGlintCentroid2( VPX_EyeType eye, VPX_RealPoint *gc )
```

Provides raw data access to the normalized centroid of the glint threshold scan in the EyeSpace window, regardless of what subsequent processing options are selected.

2.2.18. GetPupilOvalRect

```
vpxs_GetPupilOvalRect  ( VPX_RealRect *ovalRect )
vpxs_GetPupilOvalRect2  ( VPX_EyeType eye, VPX_RealRect *ovalRect )
```

Retrieves the rectangle in **RAW** EyeSpace coordinates (normalized with respect to the EyeCamer window) that specifies the oval (ellipse) fit to the pupil.
Separate rectangles are available for Eye_A, or Eye_B if binocular mode is on. Monocular ViewPoint uses Eye_A by default.

2.2.19. GetMeasuredViewingDistance

```
vpxs_GetMeasuredViewingDistance( VPX_RealType *vd )
```

Provides the physical distance (Millimeters) of the subject's eye to the display screen, as specified by the user in the ViewPoint GeometryGrid dialog.
When the head is free move and a head tracker is used, this value may not be accurate.

2.2.20. GetMeasuredScreenSize

```
vpxs_GetMeasuredScreenSize( VPX_RealPoint *displaySize )
```

Provides the physical size of the display screen (Millimeters), as calculated from the ViewPoint GeometryGrid dialog specifications.

2.2.21. GetHeadPositionAngle

```
vpxs_GetHeadPositionAngle( VPX_PositionAngle *hpa )
```

Retrieves head position and angle data. The PositionAngle structure is defined in `VPX.h`.
** available with head tracker option only **
NOTE: Subject to change.

2.2.22. GetStatus

```
Dim status as Integer
status=vpxs_GetStatus(statusIndex)
```

This provides a simple way to determine the current status or state of various ViewPoint operations. The input *statusIndex* should be a number. The number corresponding to a ViewPoint operation are as given below:

VPX_STATUS_HEAD=0;
VPX_STATUS_ViewPointIsRunning=1;
VPX_STATUS_VideoIsFrozen=2;
VPX_STATUS_DataFileIsOpen=3;
VPX_STATUS_DataFileIsPaused=4;
VPX_STATUS_AutoThresholdInProgress=5;
VPX_STATUS_CalibrationInProgress=6;
VPX_STATUS_StimulusImageShape=7;
VPX_STATUS_BinocularModeActive=8;
VPX_STATUS_SceneVideoActive=9;
VPX_STATUS_DistributorAttached=10;
VPX_STATUS_TAIL=11;