

ICONICS, Inc.
100 Foxborough Blvd.
Foxborough, MA 02035
Phone: (508) 543-8600
Fax: (508) 543-1503
Worldwide Support Centers
www.iconics.com

© ICONICS, Inc. All rights reserved.
Specifications are subject to change without notice. GENESIS, GENESIS32, Pocket GENESIS, BizViz and their respective modules, OPC-To-The-Core, and Visualize Your Enterprise are trademarks of ICONICS, Inc. Other product and company names mentioned herein may be trademarks of their respective owners.

AlarmWorX32 Indicator ActiveX OLE Automation Reference

To access the Automation interface from VB and VBA, AlarmWorX32 Indicator must be made available by choosing **Project - References** from the main menu in the VB or VBA development environment and selecting **AwxInd32 1.0 Type Library** in the list of available references.

Available Control Properties

Members of 'AwxIndicator':

BackColor As OLE_COLOR

Description

This is the "replacement" color of the Alarm Indicator. It is the color that you see behind the flashing color when there are alarms coming in. Property is read/write.

Remarks

Please note, the replacement color can only be a valid color that already exists in the bitmap, otherwise you will not see any change.

OLE_COLORs can be set in more than one way. You can use a VB color constant such as vbWhite, an RGB color value, or a Long integer.

Example

This example shows how to read and write the background color.

```
Dim backColor As Long
backColor = AwxIndicator1.BackColor
AwxIndicator1.BackColor = 16777215
'Or:
AwxIndicator1.BackColor = vbWhite
'Or:
AwxIndicator1.BackColor = RGB(255, 255, 255)
'(All examples make the background color white)
```

CanUserConfigure As Boolean

Description

True if configuration is enabled for the currently logged in user. Property is read-only.

Example

This example shows how to determine if user configuration is enabled.

MsgBox AwxIndicator1.CanUserConfigure

CanUserLauchDoc As Boolean

Description

True if launching a document is enabled for the currently logged in user. Property is read-only.

Example

This example shows how to determine if launching a document is enabled.

MsgBox AwxIndicator1.CanUserLaunchDoc

CanUserSuppressSound As Boolean

Description

True if sound suppression is enabled for the currently logged in user. Property is read-only.

Example

This example shows how to determine if sound suppression is enabled.

MsgBox AwxIndicator1.CanUserSuppressSound

Caption As String

Description

This is the text that appears when you hover over the Alarm Indicator with your mouse. Property is read/write.

Example

This example shows how to read and write the caption.

```
Dim caption As String
caption = AwxIndicator1.Caption
AwxIndicator1.Caption = “My Caption”
```

FileEnabled As Boolean

Description

True if a configuration file is enabled. Property is read/write.

Example

This example shows how to determine if a configuration file is enabled.

```
Dim bEnabled As Boolean
bEnabled = AwxIndicator1.FileEnabled
```

This example shows how to turn on/off a configuration file.

```
AwxIndicator1.FileEnabled = True
```

FileName As String

Description

This is the name of the configuration file. Property is read/write.

Remarks

Use this property to load and save configuration files. First set the property to the correct path and file name, and then call either FileLoad() or FileSave().

Example

This example shows how to read and write the configuration file name.

```
Dim fileName As String
fileName = AwxIndicator1.FileName
AwxIndicator1.FileName = "C:\My Directory\AwxIndCfg.awi"
```

FillColor As OLE_COLOR

Description

This is the "normal" color of the Alarm Indicator. It is the color that flashes when there are no alarms coming in. This can be any color. Property is read/write.

Remarks

OLE_COLORs can be set in more than one way. You can use a VB color constant such as vbWhite, an RGB color value, or a Long integer.

Example

This example shows how to read and write the fill color.

```
Dim fillColor As Long
fillColor = AwxIndicator1.FillColor
AwxIndicator1.FillColor = 16777215
'Or:
AwxIndicator1.FillColor = vbWhite
'Or:
AwxIndicator1.FillColor = RGB(255, 255, 255)
'(All make the fill color white)
```

Filter As String

GlobalSoundSuppression As Boolean

Description

True if global sound suppression is enabled. Property is read/write.

Example

This example shows how to determine if global sound suppression is enabled.

```
Dim bEnabled As Boolean
bEnabled = AwxIndicator1.GlobalSoundSuppression
```

This example shows how to turn on/off global sound suppression.

```
AwxIndicator1.GlobalSoundSuppression = True
```

IgnoreNormalUnacked As Boolean

LaunchDoc As String

Description

This is the file name of the Word document to be launched from the Alarm Indicator. Property is read/write.

Example

This example shows how to read and write the launch document file name.

```
Dim launchDoc As String
launchDoc = AwxIndicator1.LaunchDoc
AwxIndicator1.LaunchDoc = "C:\My Directory\MyWordDoc.doc"
```

LaunchDocVia As tagINVOKE_VIA

Description

This property corresponds to a constant that determines which method can be used to launch a document. Property is read/write.

Remarks

The tagINVOKE_VIA constant values are:

SingleClick = 1
DoubleClick = 2
ContextMenu = 3

Example

This example shows how to read and write the document launching method.

```
Dim n As tagINVOKE_VIA
n = AwxIndicator1.LaunchDocVia
AwxIndicator1.LaunchDocVia = ContextMenu
```

MouseIcon As IPictureDisp

Picture As IPictureDisp

Description

This is the bitmap object that is displayed on the Alarm Indicator.

Example

This example shows how to grab the Picture object.

```
Dim pic As IPictureDisp
Set pic = AwxIndicator1.Picture
```

RangeCollection As Object

Description

This object returns a RangeCollection, which is a collection of Range objects that determine the Alarm Indicator's color depending on the severity of the alarm. There is only one RangeCollection per Alarm Indicator instance; however, there can be many Range objects in each RangeCollection.

Remarks

The RangeCollection object has 3 properties and 2 methods:

```
Clone As Object
Count As Long
Item As Object

AddItem(NewItem As Object)
DeleteItem(ToDelete As Object)
```

Example

This example shows how to grab the RangeCollection object.

```
Dim rc As RangeCollection
Set rc = AwxIndicator1.RangeCollection
```

SuppressSoundPeriod As Long

Description

This specifies the number of seconds the sound will be suppressed when you invoke the "Suppress Sound" action. Property is read/write.

Example

This example shows how to read and write the sound suppression period.

```
Dim n As Long
n = AwxIndicator1.SuppressSoundPeriod
AwxIndicator1. SuppressSoundPeriod = n + 1
```

SuppressSoundVia As tagINVOKE_VIA

Description

This property corresponds to a constant that determines which method can be used to suppress a sound.

Remarks

The tagINVOKE_VIA constant values are:

SingleClick = 1
DoubleClick = 2
ContextMenu = 3

Example

This example shows how to read and write the sound suppression method.

```
Dim n As tagINVOKE_VIA
n = AwxIndicator1.SuppressSoundVia
AwxIndicator1. SuppressSoundVia = SingleClick
```

URLPath As String

Description

This specifies the URL path of a configuration file. Property is read/write.

Example

This example shows how to read and write the URL path.

```
Dim path As String
path = AwxIndicator1.URLPath
AwxIndicator1.URLPath = “http://www.iconics.com/awxindcfg.awi”
```

Members of ‘Range’:

Clone As Object

Description

This is a copy of the Range object. Property is read-only.

Example

This example shows how to grab a copy of the Range object.

```
Dim range As Range
range = AwxIndicator1.RangeCollection.Item(0).Clone
```

Color As Long

Description

This is the color that will flash when alarms within this particular severity range are coming in. Property is read/write.

Remarks

This property accepts all valid OLE_COLORs.

Example

This example shows how to read and write the range color.

```
Dim color As Long
color = AwxIndicator1.RangeCollection.Item(0).Color
AwxIndicator1.RangeCollection.Item(0).Color = 16777215
‘Or:
AwxIndicator1.RangeCollection.Item(0).Color = vbWhite
‘Or:
AwxIndicator1.RangeCollection.Item(0).Color = RGB(255, 255, 255)
‘(All make the range’s color white)
```

FlashPeriod As Long

Description

This is the duration (in milliseconds) of each flash for that particular severity range. Property is read/write.

Example

This example shows how to read and write the flash period.

```
Dim n As Long
n = AwxIndicator1.RangeCollection.Item(0).FlashPeriod
AwxIndicator1.RangeCollection.Item(0).FlashPeriod = 200
```

IsWav As Boolean

Description

Returns true if the current Range object has a valid file associated with it. Returns false if it cannot locate the file on the system.

Example

This example shows how to read the IsWav property.

```
Dim bEnabled As Boolean
bEnabled = AwxIndicator1.RangeCollection.Item(0).IsWav
```

PlayPeriod As Long

Description

This specifies the number of milliseconds between each time the .wav file is played. Property is read/write.

Remarks

If the .wav file that you are using is longer than the play period, the sound will not cut off; it will immediately restart once the .wav file is done playing.

Example

This example shows how to read and write the play period.

```
Dim n As Long
n = AwxIndicator1.RangeCollection.Item(0).PlayPeriod
AwxIndicator1.RangeCollection.Item(0).PlayPeriod = 10000
```

Severity As Long

Description

This is the severity level of the current range object. Property is read/write.

Remarks

If you change the severity level of a range to be lower than the severity level of any ranges below it, it will delete those ranges and your current range will encompass the entire severity level of the previous ranges combined. It will never create a brand new severity level.

Example

This example shows how to read and write the severity level.

```
Dim n As Long
n = AwxIndicator1.RangeCollection.Item(0).Severity
AwxIndicator1.RangeCollection.Item(0).Severity = 500
```

WavFileName As String

Description

This is a property that you set before loading or playing a .wav file for a particular range. Property is read/write.

Remarks

Use this property to load and play .wav files. First set the property to the correct path and file name, and then call either LoadWavFile() or PlayWav().

Example

This example shows how to read and write the .wav file name.

```
Dim wavFile As String
wavFile = AwxIndicator1.RangeCollection.Item(0).WavFileName
```

AwxIndicator1.RangeCollection.Item(0).WavFileName = “C:\wav_file.wav”

Members of ‘RangeCollection’:

Clone As Object

Description

This is a copy of the RangeCollection object. Property is read-only.

Example

This example shows how to grab a copy of the RangeCollection object.

```
Dim rc As RangeCollection
rc = AwxIndicator1.RangeCollection.Clone
```

Count As Long

Description

This is the number of Range objects in the RangeCollection. (Read-only).

Example

This example shows how to read the count.

```
Dim n As Long
n = AwxIndicator1.RangeCollection.Count
```

Item(Index As Long) As Object

Description

This corresponds to each Range object within the RangeCollection. Item is an array of objects. Property is read-only.

Remarks

The Item array corresponds to the Range objects in reverse order (or by decreasing severity). The first array index is 0 and it goes up to the RangeCollection Count – 1.

Example

This example shows how to grab a Range object from the RangeCollection Item array.

```
Dim r As Range
Set r = AwxIndicator1.RangeCollection.Item(0)
‘This grabs the Range object of highest severity
```

Available Control Methods

Members of 'AwxIndicator':

Sub DoContextMenu()

Description

Launches the context menu so that a user can then click on an option.

Example

This example shows how to launch the context menu.

AwxIndicator1.DoContextMenu

Sub DoLaunchDoc()

Description

Launches the document specified in the LaunchDoc property.

Example

This example shows how to perform the LaunchDoc action.

AwxIndicator1.DoLaunchDoc

Sub DoLeftMouseButton()

Description

Simulates a left mouse button click, thus performing whichever action (if any) is associated with the left mouse button.

Example

This example shows how to simulate a left mouse button click.

AwxIndicator1.DoLeftMouseButton

Sub DoLeftMouseDoubleClick()

Description

Simulates a left mouse button double-click, thus performing whichever action (if any) is associated with double-clicking the left mouse button.

Example

This example shows how to simulate a left mouse button double-click.

AwxIndicator1.DoLeftMouseDoubleClick

Sub DoSuppressSound()

Description

Performs the sound suppression action. This will silence the Alarm Indicator for the number of seconds specified in the SoundSuppressionPeriod property.

Example

This example shows how to suppress the sound.

```
AwxIndicator1.DoSuppressSound
```

Sub FileLoad()

Description

Loads the configuration file that is currently in the FileName property.

Remarks

You must set the FileName property and make sure that FileEnabled is set to "True" before calling this method or it will have no effect.

Example

This example shows how to load a configuration file.

```
AwxIndicator1.FileName = "C:\My Directory\AwxIndCfg.awi"
AwxIndicator1.FileEnabled = True
AwxIndicator1.FileLoad
'Loads the file AwxIndCfg.awi
```

Sub FileSave()

Description

Saves the configuration file that is currently in the FileName property.

Remarks

You must set the FileName property and make sure that FileEnabled is set to "True" before calling this method or it will have no effect.

Example

This example shows how to save a configuration file.

```
AwxIndicator1.FileName = "C:\My Directory\AwxIndCfg1.awi"
AwxIndicator1.FileEnabled = True
AwxIndicator1.FileSave
'Saves the current configuration out to the file AwxIndCfg1.awi
```

Sub ForceConfigMode(Force As Boolean)

Description

Allows the Alarm Indicator to exit runtime and enter configuration mode while the container remains in runtime mode.

Example

This example shows how to force configuration mode.

AwxIndicator1.ForceConfigMode(True)

GetAlarmOleBlob() As Long

Sub SetAlarmOleBlob(pBlob As Long)

Description

These above two methods are part of AlarmOLE.

ReplaceFilePath(OldSubstring As String, NewSubstring As String) As Long

Description

Works over path name parameters of pick dynamics. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces file path in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceFilePath("C:\Windows\Temp",
"D:\Temp")
'if Status <> 0 then there was no replacement performed or
an error occurs
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

ReplaceHost(OldHostName As String, NewHostName As String) As Long

Description

Works over both data source tags and path name attributes and replaces node name **substring** within URL path only. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces host name in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceHost("Host1", "Host2")
'if Status <> 0 then there was no replacement performed or
an error occurs
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

ReplaceHostEx(OldHostNameSubstring As String, NewHostNameSubstring As String, MatchCase As Long, MatchWholeWord As Long) As Long

Description

Works over both data source tags and path name attributes and replaces node name **substring** within URL path only, and supports case-sensitivity, wildcard strings and MatchWholeWord flag. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces host name in whole display, regarding "case" and
"whole words" options

Dim Status As Long
Status = ThisDisplay.ReplaceHostEx("Host1", "Host2", True,
True)
'if Status <> 0 then there was no replacement performed or
an error occurs
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

ReplaceStringInString(StingToReplace As String, OldSubstring As String, NewSubstring As String) As String

Description

Replaces **OldSubstring** with **NewSubstring** inside (within) StringToReplace. Returns 0 on success and HRESULT when something fails.

ReplaceTag(OldSubstring As String, NewSubstring As String) As Long

Description

Works over data source tags in dynamic actions of all display objects. Replaces **OldSubstring** with **NewSubstring** in certain situations. Returns 0 on success and HRESULT when something fails.

Typical use of this method:

Example

```
'VBA example, works only in configure mode
'replaces tag name in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceTag("gfwsim.ramp.float",
"gfwsim.sine.double")
'if Status <> 0 then there was no replacement
'performed or an error occurs
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

Members of 'Range':

Sub LoadWavFile()

Description

Sets the .wav file to be played when that specific range is in alarm condition. Uses the .wav file that is currently residing in the WavFileName property.

Remarks

You must set the WavFileName property before calling this method or it may return an error.

Example

This example shows how to load a .wav file.

```
ThisDisplay.AwxIndicator1.RangeCollection.Item(0).WavFileName =
"C:\Program Files\ICONICS\GENESIS32\Bin\MMXSounds\DRUMS.WAV"
ThisDisplay.AwxIndicator1.RangeCollection.Item(0).LoadWavFile
ThisDisplay.AwxIndicator1.RangeCollection.Item(0).PlayWav
```

Sub PlayWav()

Description

Plays the .wav file that is currently residing in the WavFileName property.

Remarks

You must set the WavFileName property before calling this method or it may return an error.

Example

This example shows how to play a .wav file.

```
ThisDisplay.AwxIndicator1.RangeCollection.Item(0).WavFileName =
"C:\Program Files\ICONICS\GENESIS32\Bin\MMXSounds\DRUMS.WAV"
ThisDisplay.AwxIndicator1.RangeCollection.Item(0).LoadWavFile
ThisDisplay.AwxIndicator1.RangeCollection.Item(0).PlayWav
```

Members of 'RangeCollection':

Sub AddItem(NewItem As Object)

Description

Adds a new item into the RangeCollection of the Alarm Indicator.

Remarks

The easiest way to do this is to clone an already existing Range object and then modify its properties to fit your new Range needs.

Example

This example shows how to clone an existing Range object and add a modified version of it to the RangeCollection.

```
Dim newRange As Range
Set newRange = ThisDisplay.AwxIndicator1.RangeCollection.Item(0).Clone
newRange.Color = vbBlue
newRange.FlashPeriod = 300
newRange.PlayPeriod = 6000
newRange.Severity = 500
ThisDisplay.AwxIndicator1.RangeCollection.AddItem newRange
ThisDisplay.AwxIndicator1.RangeCollection.DeleteItem newRange
```

Sub DeleteItem(ToDelete As Object)

Description

Deletes an item from the RangeCollection of the Alarm Indicator.

Example

```
Dim newRange As Range
Set newRange = ThisDisplay.AwxIndicator1.RangeCollection.Item(0).Clone
newRange.Color = vbBlue
newRange.FlashPeriod = 300
newRange.PlayPeriod = 6000
newRange.Severity = 500
ThisDisplay.AwxIndicator1.RangeCollection.AddItem newRange
ThisDisplay.AwxIndicator1.RangeCollection.DeleteItem newRange
```

AlarmWorX32 Logger OLE Automation Reference

This reference describes the OLE Automation features available in the AlarmWorX32 Logger.

The AlarmWorX32 logger provides a COM interface that allows external programs to manipulate the logger as it is running. The interface is available to all programming languages that support COM including Visual Basic (VB), Visual Basic for Applications (VBA) and Microsoft Visual C^{++}.

To access the Automation interface from VB and VBA, AlarmWorX32 logger must be made available by choosing **Project - References** from the main menu in the VB or VBA development environment and selecting **AWXLog32 Type Library** from the list of available references.

Available Control Properties

LoggingEnabled As Boolean

Description

True if logging to database is enabled. Applies to all running configurations that match the current **Filter Property**. Property is read/write.

Remarks

Values written to this property are transient. The value written is not stored in the configuration database and hence is "lost" when the logger is shut down.

Reading this property will throw an error in the following cases:

The logger has just started and the configuration has not been read yet.

Multiple running configurations match the filter and they have differing values (some True, some False).

Example

This example shows how to determine if logging is enabled.

```
Dim bEnabled as Boolean
bEnabled = AWXLogger.LoggingEnabled
```

This example shows how to turn on/off logging

```
AWXLogger.LoggingEnabled = True
```

PrintingEnabled As Boolean

Description

True if logging to printer is enabled. Applies to all running configurations that match the current **Filter Property**. Property is read/write.

Remarks

Values written to this property are transient. The value written is not stored in the configuration database and hence is "lost" when the logger is shut down.

Reading this property will throw an error in the following cases:

The logger has just started and the configuration has not been read yet.

Multiple running configurations match the filter and they have differing values (some True, some False).

Example

This example shows how to determine if printing is enabled.

```
Dim bEnabled as Boolean
bEnabled = AWXLogger.PrintingEnabled
```

This example shows how to turn on/off printing.

```
AWXLogger.PrintingEnabled = True
```

Count As Long

Description

Returns the number of running configurations whose names match the current Filter Property. Property is read-only.

Remarks

Reading this property will throw an error if the logger has just started and the configuration has not been read yet.

Example

```
Dim n as Long
n = AWXLogger.Count
```

Filter As String

Description

Wildcard string that a running configuration name must match in order to be affected by other methods and properties of this object. String property is read/write.

Remarks

If the name of a running configuration does ***not*** match the filter, then that configuration is ignored by all other methods and properties of this object. The default initial value for this Property is "*" meaning all running configurations will match, hence by default, all of the methods and properties will operate simultaneously on all running configurations.

Example

```
Dim sFilter as String
```

```
sFilter = AWXLogger.Filter

'set filter for all configurations that begin with b
AWXLogger.Filter = "b*"
```

Item(Index As Long) As String

Description

The name of a running configuration (that matches the **Filter Property**) for a given index value in the array (VB Collection). String property is read-only.

Syntax

Item(long n)

Where **n** is an integer from 1 to **Count**

Remarks

The AlarmWorX32 logger Automation object is designed as a VB Collection. Each element in the collection is the name of an active configuration that matches the current **Filter Property.**

Example

These three examples do the same thing -- print the name of each configuration.

```
Dim AWXLogger As AWXLog32Auto
Set AWXLogger = New AWXLog32Auto

' method 1
For Each ConfigName In AWXLogger
   Print ConfigName
Next ConfigName

' method 2
For i = 1 To AWXLogger.Count
   Print AWXLogger(i)
Next i

' method 3
For i = 1 To AWXLogger.Count
   Print AWXLogger.Item(i)
Next i
```

AlarmWorX32 Report ActiveX OLE Automation Reference

This reference describes the OLE Automation features available in the AlarmWorX32 Report.

Automation Interfaces

The AlarmWorX32 Report provides a COM interface that allows automation interfaces run from within the ActiveX container to manipulate the Report ActiveX control as it is running. The interface is available to all programming languages that support COM, including Visual Basic (VB), Visual Basic for Applications (VBA), and Microsoft Visual C^{++}.

To access the Automation interface from VB and VBA, AlarmWorX32 Report must be made available by choosing **Project > Components** from the main menu in the VB or VBA development environment and selecting **ICONICS AWXRep32 ActiveX** in the list of available components.

The description of every property or method in this reference is followed by the code example for typical use of that property or method. The example code is written in Visual Basic for Applications (VBA) language, built in the AlarmWorX32 container application (Awx32.exe). It assumes that AWXRep32 ActiveX is embedded within **AlarmWorX** display and its name property is set to "AWXRep321."

Available Control Properties

AWXRep32Title

Type: String

Description

Sets/gets title string on General property page.

Example

```
Dim strNewTitle as String
ThisDocument.AWXRep321.AWXRep32Title=strNewTitle
```

AppearanceMode

Type: Long

Description

Determines the appearance of data grid. This property can be one of the following:

0 - flat look
1 - 3D look

Example

Sets 3D look for the grid.

```
ThisDocument.AWXRep321.AppearanceMode=1
```

BackColor

Type: OLE_COLOR

Description

Sets/gets grid background color.

Example

Sets grid background color to cyan.

```
ThisDocument.AWXRep321.BackColor=RGB(0,255,255)
```

BorderType

Type: String

Description

Sets/gets control's border appearance. Can be one of the following:

"0" - none

"1" - flat border

"2" - 3D border

Example

Sets 3D look for the control's border.

```
ThisDocument.AWXRep321.BorderType="2"
```

ChartBackColor

Type: OLE_COLOR

Description

Sets/gets summary chart background color.

Example

Sets summary chart background color to white.

```
ThisDocument.AWXRep321.ChartBackColor=RGB(255,255,255)
```

ChartLegendHeight

Type: Short

Description

These properties determine the dimensions of the corresponding chart element. The width and height of a particular chart element are given in percents of the total width or height of the chart area.

ChartLegendLeft

ChartLegendTop

Type: Short

Description

These properties determine the location of the corresponding chart element within the chart area. The coordinates are given in percents of the total width or height of the chart area, assuming that the coordinate origin (point {0, 0}) is located at the left top corner of the chart area.

ChartLegendVisible

Type: Boolean

Description

These properties set/get the visibility of chart legend and/or title. Setting either property to FALSE will hide the corresponding chart element.

Default value: TRUE

ChartLegendWidth

Type: Short

Description

These properties determine the dimensions of the corresponding chart element. The width and height of a particular chart element are given in percents of the total width or height of the chart area.

ChartPlotHeight

Type: Short

Description

These properties determine the dimensions of the corresponding chart element. The width and height of a particular chart element are given in percents of the total width or height of the chart area.

ChartPlotLeft

ChartPlotTop

Type: Short

Description

These properties determine the location of the corresponding chart element within the chart area. The coordinates are given in percents of the total width or height of the chart area, assuming that the coordinate origin (point {0, 0}) is located at the left top corner of the chart area.

ChartPlotWidth

Type: Short

Description

These properties determine the dimensions of the corresponding chart element. The width and height of a particular chart element are given in percents of the total width or height of the chart area.

ChartTitle

Type: String

Description

Sets/gets a string value that appears in the summary chart title area.

Example

```
ThisDocument.AWXRep321.ChartTitle="Alarm Summary By Tags"
```

ChartTitleHeight

Type: Short

Description

These properties determine the dimensions of the corresponding chart element. The width and height of a particular chart element are given in percents of the total width or height of the chart area.

ChartTitleLeft

ChartTitleTop

Type: Short

Description

These properties determine the location of the corresponding chart element within the chart area. The coordinates are given in percents of the total width or height of the chart area, assuming that the coordinate origin (point {0, 0}) is located at the left top corner of the chart area.

ChartTitleUpdateOnDataSelect

Type: Boolean

Description

Sets/gets the value of a flag that determines whether the chart title is updated when user clicks on a data series (either bar or pie segment) inside the plot. If set to TRUE (default), the name of selected data series is attached to the chart title.

ChartTitleVisible

Type: Boolean

Description

These properties set/get the visibility of chart legend and/or title. Setting either property to FALSE will hide the corresponding chart element.

Default value: TRUE

ChartTitleWidth

Type: Short

Description

These properties determine the dimensions of the corresponding chart element. The width and height of a particular chart element are given in percents of the total width or height of the chart area.

ChartType

Type: Integer

Description

Sets/gets the type of alarm summary chart. Can be one of the following:

0 - bar type

1 - pie type

Example

Sets summary chart to pie type.

```
ThisDocument.AWXRep321.ChartType=1
```

Columns

Type: String

Description

Sets/gets comma-separated list of database columns to be shown on grid.

Example

This line configures control to show only EventTime, Tag and Message columns.

```
ThisDocument.AWXRep321.Columns="EventTime, Tag, Message"
```

Remarks

If database column name contains one or more space characters, the column name should be enclosed in square brackets, as in this example:

```
ThisDocument.AWXRep321.Columns="[Column 1], [Column 2]"
```

ConnectionString

Type: String

Description

Sets/gets ADO connection string used by AWXReport32 control to access data.

Example

```
Dim strConnString as String
strConnString="Provider=Microsoft.Jet.OLEDB.4.0; _
 Data Source=c:\alarms\AWXLog32.mdb; _
 User ID=Admin; Password=SYSTEM"
ThisDocument.AWXRep321. ConnectionString = strConnString
```

DisplayActiveFilterName

Type: Boolean

Description

Sets/gets Boolean value that controls whether the active filter(s) name(s) is(are) shown on the grid title bar.

Example

```
ThisDocument.AWXRep321. DisplayActiveFilterName = TRUE
```

DisplayGridTitle

Type: Boolean

Description

Sets/gets Boolean value that controls the visibility of the grid title bar.

Example

```
ThisDocument.AWXRep321. DisplayGridTitle = TRUE
```

EnableToolbar

Type: Boolean

Description

Setting this property to FALSE prohibits the floating toolbar to pop up when user

double-clicks inside the control.

Default value - TRUE.

Example

To disable tools

```
ThisDocument.AWXRep321.EnableTools=FALSE
```

FilteringString

Type: String

Description

Sets/gets SQL WHERE statement that determines filtering of alarm data.

Example

These statements configure control to show only Humidity alarms or Pressure alarms.

```
Dim strFiltrStmt as String
strFiltrStmt="WHERE  (Tag='Humidity' OR Tag='Pressure') "
ThisDocument.AWXRep321. FilteringString = strFiltrStmt
```

Font

Type: StdFont

Description

Returns a reference to the control's default font.

Example

To set control's default font to Arial, 14 pt., use the following statements:

```
With ThisDocument.AWXRep321.Font
.Name="Arial"
.Size=14
End With
```

ForeColor

Type: OLE_COLOR

Description

Sets/gets Forecolor property of data grid.

Example

```
ThisDocument.AWXRep321. ForeColor=RGB(0,0,0)
```

GridBackColor

Type: OLE_COLOR

Description
Sets/gets color value used to paint the gaps between rightmost column and right edge of the grid, last row and bottom edge of the grid, and row headers column (the 'back area' of the grid).

Example

```
ThisDocument.AWXRep321. GridBackColor =RGB(0,127,0)
```

GridBorderStyle

Type: Long

Description

Sets/gets the style of data grid border. Can be one of the following:

0 - no border
1 - fixed single border style

Example

```
ThisDocument.AWXRep321.GridBorderStyle=1
```

GridDefRowHeight

Type: Integer

Description
Sets/gets the default row height for data grid.

Example

```
ThisDocument.AWXRep321. GridDefRowHeight=16
```

GridHorizLinesColor and GridVertLinesColor

Type: OLE_COLOR

Description
Sets/gets color for horizontal or vertical grid lines.

Example

```
ThisDocument.AWXRep321. GridHorizLinesColor
=RGB(127,127,127)
ThisDocument.AWXRep321. GridVertLinesColor =
RGB(127,127,127)
```

GridHorizLinesStyle and GridVertLinesStyle

Type: Integer

Description

Sets/gets style for horizontal or vertical grid lines. Can be one of the following:

0 - no line

1 - solid line

2 - dot line

3 - dashdot line

4 - dashdotdot line

Example

```
ThisDocument.AWXRep321. GridHorizLinesStyle =0
ThisDocument.AWXRep321. GridVertLinesStyle = 2
```

GridHorizLinesWidth and GridVertLinesWidth

Type: Integer

Description

Sets/gets the widths for the horizontal or vertical grid lines when corresponding grid line style (property **GridHorizLinesStyle** or **GridVertLinesStyle**) is set to 1 - solid line. For all other line styles, these properties must be set to 1.

Example

Sets both horizontal and vertical grid lines to solid line style, 2 pixels wide.

```
ThisDocument.AWXRep321. GridHorizLinesStyle =1
ThisDocument.AWXRep321. GridVertLinesStyle = 1
ThisDocument.AWXRep321. GridHorizLinesWidth =2
ThisDocument.AWXRep321. GridVertLinesWidth = 2
```

GridTitleBackColor

Type: OLE_COLOR

Description

Sets/gets the background color for the grid's title bar.

Example

Sets title bar background color to dark blue.

```
ThisDocument.AWXRep321. GridTitleBackColor =GRB(0,0,127)
```

GridTitleTextColor

Type: OLE_COLOR

Description

Sets/gets the text color for the grid's title bar.

Example

Sets grid's title bar text color to yellow.

```
ThisDocument.AWXRep321. GridTitleBackColor
=GRB(255,255,0)
```

HeaderColor

Type: OLE_COLOR

Description

Sets/gets the background color for grid's column headers.

Example

Sets color of grid's column header to dark green.

```
ThisDocument.AWXRep321. GridTitleBackColor =GRB(0,127,0)
```

IsConnected

Type: Boolean

Description

This is read-only property, indicating that control is connected to the database.

Example

```
If  ThisDocument.AWXRep321.IsConnected Then
 ``` do something here  ```
End If
```

NoSorting

Type: Boolean

Description

Sets/gets **Boolean** value that indicates whether records sorting has been enabled or not. By default, this property is set to TRUE (sorting disabled), providing best data retrieval performance.
```

Example

To enable sorting:
```
ThisDocument.AWXRep321.NoSorting=FALSE
```

QueryText

Type: String

Description

Sets/gets the SQL statement used to populate data grid in detailed report mode.

Example

```
Dim strQry as String
strQry=ThisDocument.AWXRep321.QueryText
MsgBox strQry
```

Remarks

Though this property is read/write, it is not recommended to set **QueryText** directly. Instead, you can set the **Columns**, **FilteringString**, and **SortingString** properties, then the value of **QueryText** will be automatically updated.

QueryTextSummary

Type: String

Description

Sets/gets the SQL statement used to populate data grid in alarm summary mode.

Example

```
Dim strQrySmr as String
strQrySmr=ThisDocument.AWXRep321.QueryTextSummary
MsgBox strQrySmr
```

Remarks

Though this property is read/write, it is not recommended to set **QueryTextSummary** directly. Instead, you can set the **SummaryColumn**, **FilteringString**, **SortingString,** or **SummaryFieldSortDir** properties. The value of **QueryTextSummary** will then be automatically updated.

ReportFileAlarms

FileAlarmsCondensed
FileAlarmsEvents
ReportFileEvents
ReportFileEventsCondensed

These properties should be considered obsolete as of version 6.1. They are supported only for backwards compatibility.

RowDividerStyle

Type: String

Description

Sets/gets the appearance of row divider on data grid (horizontal grid lines). Can be one of the following:

0 - No dividers

1 - Black line

2 - Dark gray line

3 - Raised

4 - Inset

5 - Use ForeColor

6 - Light gray line

This property can be used to set the appearance of both vertical and horizontal grid lines in one statement. The same can be done using the **GridHorizLineColor** and **GridVertLinesColor** properties independently.

Example

To set Black row divider:

```
ThisDocument.AWXRep321.RowDividerStyle=1
```

ShowSummaryChart

Type: Boolean

Description

Sets/gets the **Boolean** value that controls how alarm records are reported. Use this property to toggle between **Chart** and **Grid Display** modes.

Examples

This statement brings control into **Chart Display** mode:

```
ThisDocument.AWXRep321. ShowSummaryChart=TRUE
```

This statement brings control into **Grid Display** mode:

```
ThisDocument.AWXRep321. ShowSummaryChart=FALSE
```

ShowSummaryGrid

Type: Boolean

Description

Sets/gets the **Boolean** value that controls how alarm records are reported in **Grid Display** mode. Use this property to toggle between **Detailed** and **Summary** views in **Grid Display** mode.

Remarks

If control is in the **Chart** mode, setting **ShowSummaryGrid** property will not have immediate effect. Only after **ShowSummaryChart** property is switched to FALSE (**Grid** mode), will the desired grid view be displayed.

Examples

These statements bring the control into **Detailed Grid Display** mode:

```
ThisDocument.AWXRep321. ShowSummaryChart =FALSE
ThisDocument.AWXRep321. ShowSummaryGrid =FALSE
```

This statement brings the control into **Summary Grid Display** mode:

```
ThisDocument.AWXRep321. ShowSummaryGrid =TRUE
```

SortByDblClick

Type: Boolean

Description

Sets/gets **Boolean** value indicating that records will be sorted by values in the particular grid's column when user double-clicks on that column's header. Subsequent double clicks on the same column header will reverse sorting direction.

Remarks

The sorting action will occur only if the **NoSorting** property is set to FALSE.

Example

```
ThisDocument.AWXRep321. NoSorting =FALSE
```

```
ThisDocument.AWXRep321. SortByDblClick=TRUE
```

SortingString

Type: String

Description

Sets/gets SQL ORDER BY statement that determines sorting of alarm data.

Example

Configures control to sort alarm data by EventTime in ascending order.

```
Dim strSortStmt as String
strSortStmt="ORDER BY EventTime ASC "
ThisDocument.AWXRep321. SortingString = strSortStmt
```

SummaryColumn

Type: String

Description

Sets/gets database column name, by which the alarm records are grouped for **Summary Grid** view and **Summary Chart.** By default, this is "[Tag]" column.

Example

This statement changes the database column to summarize records by "Area":

```
ThisDocument.AWXRep321.SummaryColumn="[Area]"
```

SummaryFieldSortDir

Type: Integer

Description

Sets/gets sort direction for the **Summary** field on the **Summary Grid** view or **Summary Chart.** Can be one of the following:
0 - ascending
1 - descending

Example

This statement changes the **Summary** field sort direction to descending:

```
ThisDocument.AWXRep321. SummaryFieldSortDir=1
```

TableName

Type: String

Description

Sets/gets name of the current database table.

Example

```
ThisDocument.AWXRep321.TableName="EventLog"
```

ToolsVisible

Type: Boolean

Description

Sets/gets visibility of toolbar.

Example

To show toolbar:

```
ThisDocument.AWXRep321.ToolsVisible=TRUE
```

ToolXPos

Type: Integer

Description

Sets/gets the X coordinate of toolbar's left top corner (in device units).

Example

```
ThisDocument.AWXRep321.ToolXPos=255
```

ToolYPos

Type: Integer

Description

Sets/gets the Y coordinate of toolbar's left top corner (in device units).

Example

```
ThisDocument.AWXRep321.ToolYPos=100
```

AutoUpdateInterval

Type: Short

Description

Sets/gets the number of seconds between automatic updates.

Example

ThisDocument.AWXRep321.AutoUpdateInterval=60

BlinkOnUpdate

Type: Boolean

Description

Sets/gets whether or not the Report ActiveX will blink on update.

Example

ThisDocument.AWXRep321.BlinkOnUpdate=True

ChartGlobalScaling

Type: Boolean

Description

Enables/disables global scaling of the chart.

Example

ThisDocument.AWXRep321.ChartGlobalScaling=True

ChartScrolling

Type: Boolean

Description

Enables/disables chart scrolling.

Example

ThisDocument.AWXRep321.ChartScrolling=True

ChartScrollSize

Type: Short

Description

Indicates how many items you can scroll through on the chart at one time.

Example

ThisDocument.AWXRep321.ChartScrollSize=10

ChartTruncateNumber

Type: Short

Description

Sets/gets the size at which long strings are truncated. Must have ChartTruncateStrings property set to “True” in order for this to take effect.

Example

ThisDocument.AWXRep321.ChartTruncateNumber=8

ChartTruncateStrings

Type: Boolean

Description

Enables/disable the truncation of strings longer than the ChartTruncateNumber.

Example

ThisDocument.AWXRep321.ChartTruncateStrings=True

EnableAutoUpdate

Type: Boolean

Description

Enables/disables automatic updates.

Example

ThisDocument.AWXRep321.EnableAutoUpdate=True

EnableWebAccess;

Type: Boolean

Description

Enables/disables Report ActiveX web access.

Example

ThisDocument.AWXRep321.EnableWebAccess=True

FileName

Type: BSTR

Description

Sets/gets the name of the Report ActiveX configuration file.

Example

ThisDocument.AWXRep321.FileName=”C:\AwxRepConfig.awr”

MaxRecords

Type: Long

Description

Sets/gets the maximum number of records that the Report ActiveX will fit.

Example

ThisDocument.AWXRep321.MaxRecords=1024

RemoteDataServer

Type: BSTR

Description

This is the Internet Information Server for remote data access.

Example

ThisDocument.AWXRep321.RemoteDataServer = “http://localhost”

URLPathName

Type: BSTR

Description

Sets/gets the URL path for a configuration file over the web. Must set URLPathUsed to “True” for this to take effect.

Example

ThisDocument.AWXRep321.URLPathName=
”http://www.iconics.com/awxrepcfg.awr”

URLPathUsed

Type: Boolean

Description

Enables/disables the use of a URL path.

Example

ThisDocument.AWXRep321.URLPathUsed=True

UseLocalSettings;

Type: Boolean

Description

Tells a URL-enabled Report ActiveX whether or not to use local settings.

Example

ThisDocument.AWXRep321.UseLocalSettings=True

HeadShadeType

Type: long

Description

Sets/gets shading type for the grid column headers and grid title bar:

0 – Soft bump

1 – Diagonal shade

2 – Horizontal shade

3 – Vertical shade

4 – Horizontal bump

5 – Vertical bump

6 – Metal

500 – None



Example

```
Dim lHeadShadeType as long
lHeadShadeType = 3
ThisDocument.AWXRep321.HeadShadeType=lHeadShadeType
```

RoundedHead

Type: boolean

Description

Controls the “roundness” of grid header buttons and title bar.

Example

```
ThisDocument.AWXRep321RoundedHead=True
```

Available Control Methods

AboutBox

Shows **About** dialog box.

Example

```
ThisDocument.AWXRep321.AboutBox
```

Connect As Boolean

Forces the control to establish connection to database, using the information set in **ConnectionString** and **TableName.** If control is already connected, the current connection will be closed and a new connection will be established. This method can be used in VBA scripts automatically to switch the report ActiveX between different databases.

Return Value

On success, returns TRUE; if for any reason connection cannot be established, returns FALSE.

Example

```
Dim strConnString as String
strConnString="Provider=Microsoft.Jet.OLEDB.4.0; _
   Data Source=c:\alarms\SomeOtherDatabase.mdb; _
     User ID=Admin; Password=SYSTEM"
ThisDocument.AWXRep321. ConnectionString = strConnString
ThisDocument.AWXRep321.TableName="SomeOtherTable"
ThisDocument.AWXRep321.Connect
      ThisDocument.AWXRep321.Refresh
```

GetColumnHeaderText(iColIndex As Integer) As String

Returns the header text for the grid column specified by iColIndex argument. The iColIndex argument is 1 - based index of the column, counting from the left of the grid. In **Summary Grid** view, iColIndex can be either 1 or 2; in **Detailed Grid** view, it can be from 1 to the maximum number of entries in the **Columns** property

Example

```
Dim strHeader1
      strHeader1=ThisDocument.AWXRep321.GetColumnHeaderT
ext(1)
```

GetColumnWidth(iColIndex As Integer) As Integer

Returns the width of the column specified by iColIndex argument. The iColIndex argument is 1 - based index of the column, counting from the left of the grid. In **Summary Grid** view, iColIndex can be either 1 or 2; in **Detailed Grid** view, it can be from 1 to the maximum number of entries in the **Columns** property. The width is given in logical units, depending on the selected font.

Example

Retrieves the current width on the leftmost column:

```
Dim cx1 As Integer
      cx1=ThisDocument.AWXRep321.GetColumnWidth(1)
```

LaunchReport(iTemplateIndex As Integer) As Integer

Creates Crystal Report based on the template specified by iTemplateIndex parameter. The value of argument iTemplateIndex can be 0 through 4 (there might be up to 5 different Crystal Report templates).

Return value

On success, returns 0; if a template for certain index does not exist or is not configured (in which case the **Crystal Report** button on the toolbar at the position corresponding to that index, counting from left, is grayed), return value is (-1).

Example

Creates Crystal Report based on the third template from the left, as seen on the **Reporter's** toolbar:

```
If  ThisDocument.AWXRep321.LaunchReport(2)=-1 Then
            MsgBox " It looks like the template
specified is not configured"
End If
```

Print

Description

Prints detailed alarm data, alarm summary, data or alarm summary chart depending on the current state of AWXReport32 control.

Example

```
      ThisDocument.AWXRep321.Print
```

Refresh

Description

Updates alarm data by requerying the database.

Example

```
ThisDocument.AWXRep321.Refresh
```

SetColumnHeaderText(iColIndex As Integer, strNewText As String)

Sets the header text for the grid column specified by iColIndex argument. The iColIndex argument is 1 - based index of the column, counting from the left of the grid. In **Summary Grid** view, iColIndex can be either 1 or 2; in **Detailed Grid** view, it can be from 1 to the maximum number of entries in the **Columns** property.

Example

```
ThisDocument.AWXRep321.SetColumnHeaderText(1,"Colu
mn1")
```

SetColumnWidth(iColIndex As Integer, iNewWidth As Integer)

Sets the width of the column specified by iColIndex argument to iNewWidth. The iColIndex argument is 1 - based index of the column, counting from the left of the grid. In **Summary Grid** view, iColIndex can be either 1 or 2; in **Detailed Grid** view, it can be from 1 to the maximum number of entries in the **Columns** property. The width is given in logical units, depending on the selected font.

Example

Doubles the current width on the leftmost column:

```
Dim cx1 As Integer
cx1=ThisDocument.AWXRep321.GetColumnWidth(1)
      ThisDocument.AWXRep321.SetColumnWidth(1, 2*cx1)
```

LoadConfigFile(BSTR newVal) as Boolean

Loads the configuration file specified in newVal.

Example

```
Dim ret As Boolean
ret = AWXRep321.LoadConfigFile("C:\My Configs\cfg.awr")
```

SaveConfigFile(BSTR newVal) as Boolean

Saves your current Report ActiveX configuration to the file specified in newVal.

Example
```
Dim ret As Boolean
ret = AWXRep321.SaveConfigFile("C:\My Configs\cfg.awr")
```

LoadConfigURL(BSTR newVal) as Boolean

Loads the configuration file specified in the URL string newVal.

Example
```
Dim ret As Boolean
ret =
AWXRep321.LoadConfigURL("http://localhost/cfg.awr")
```

SetDBType(short NewDBType)

Sets the database type.

Example
```
AWXRep321.SetDBType(1)
```

GetColumnsNumber() as Short

Returns the number of columns in your current report configuration.

Example
```
MsgBox AWXRep321.GetColumnsNumber
```

GetFiltersNumber() as Short

Returns the number of filters currently available to your report.

Example
```
MsgBox AWXRep321.GetFiltersNumber
```

GetSortEntriesNumber() as Short

Returns the number of columns on which your report is currently sorting.

Example
```
MsgBox AWXRep321.GetSortEntriesNumber
```

GetColumn(short iIndex) as OGridColumnWrapper

Gets the Column object corresponding to iIndex. Returns "Nothing" if the index is invalid. Column index numbers start at 1.

Example
```
Dim col As OGridColumnWrapper
Set col = AWXRep321.GetColumn(1)
```

AddColumn(BSTR newCol) as OGridColumnWrapper

Adds the specified column to your report. Returns "Nothing" if argument is not a valid column name. You can then modify the column through the OGridColumnWrapper automation properties and methods.

Example

```
Dim col As OGridColumnWrapper
Set col = AWXRep321.AddColumn("Severity")
```

SwitchColumns(short iIndex1, short iIndex2)

Switches the display order of the two columns corresponding to the specified indices. Returns "Nothing" if the column index is invalid.

Example

```
AWXRep321.SwitchColumns(1,2)
```

DeleteColumn(short iIndex)

Deletes the column corresponding to the specified index. Returns "Nothing" if the column index is invalid.

Example

```
AWXRep321.DeleteColumn(2)
```

GetSummaryColumn(short iIndex) as OGridSummaryColumnWrapper

Gets the OGridSummaryColumnWrapper object corresponding to the specified column index. The summary column is the header associated with each column. Returns "Nothing" if the column index is invalid.

Example

```
Dim sumcol As OGridSummaryColumnWrapper
Set sumcol = AWXRep321.GetSummaryColumn(1)
```

UpdateColumnsInfo()

Forces a refresh of the column information.

Example

```
AWXRep321.UpdateColumnsInfo
```

GetFilter(short iIndex) as OFilterWrapper

Gets the filter corresponding to the specified index.

Example

```
Dim filter As OFilterWrapper
Set filter = AWXRep321.GetFilter(1)
```

AddFilter(BSTR newCol) as OFilterWrapper

Adds a filter on the specified column. You can then modify the filter through the OFilterWrapper automation properties and methods.

Example

```
Dim filter As OFilterWrapper
Set filter = AWXRep321.AddFilter("Severity")
```

DeleteFilter(short iIndex)

Deletes the specified filter.

Example

```
AWXRep321.DeleteFilter(1)
```

UpdateFiltersInfo()

Forces an update of the filter information.

Example

```
AWXRep321.UpdateFiltersInfo
```

GetSortEntry(short iIndex) as OSortEntryWrapper

Returns an OSortEntryWrapper object corresponding to the specified index. Returns "Nothing" if the index is invalid.

Example

```
Dim sort As OSortEntryWrapper
Set sort = AWXRep321.GetSortEntry(1)
```

AddSortEntry(BSTR newCol) as OSortEntryWrapper

Adds a sort for the specified column. You can then modify the sort entry through the OSortEntryWrapper automation properties and methods.

Example

```
Dim sort As OSortEntryWrapper
Set sort = AWXRep321.AddSortEntry("Tag")
```

DeleteSortEntry(short iIndex)

Deletes the sort entry corresponding to the specified index.

Example

```
AWXRep321.DeleteSortEntry(1)
```

UpdateSortInfo()

Forces an update of the sort entry information.

Example
```
AWXRep321.UpdateSortInfo
```

SwitchSortEntries(short iIndex1, short iIndex2)

GetCellText(integer iColumnIndex, long iRowIndex) as VARIANT

Returns the contents of the cell specified by the grid coordinates.

Example
```
Dim cell As Variant
cell = AWXRep.GetCellText(1,1)
MsgBox cell
```

GetGridRowCount() as VARIANT

Returns the number of rows in the grid. Returns –1 if there are no records or the database is not yet connected.

Example
```
Msgbox AWXRep321.GetGridRowCount
```

GetSelectedRowIndex() as VARIANT

Returns the index of the row that is currently selected. Returns –1 if no cell is selected.

Example
```
MsgBox AWXRep321.GetSelectedRowIndex
```

GetSelectedColIndex() as VARIANT

Returns the index of the column that is currently selected. Returns –1 if no cell is selected.

Example
```
MsgBox AWXRep321.GetSelectedColIndex
```

PrintToDefault(numOfCopies as Integer, orientation as Integer) as Long

Sends the current report to the default printer. Use 0 for portrait orientation and 1 for landscape.

Example
```
AWXRep321.PrintToDefault(1,0)
```

PrintToDevice(BSTR strDriverName, BSTR strDeviceName,BSTR strPortName, numberOfCopies as Integer, orientation as Integer) as Long

Prints to the specified device.

CanNavigateNextTable() as Boolean

Returns whether or not the report can navigate to the next table. Returns false if there are no other tables.

Example

```
MsgBox AWXRep321.CanNavigateNextTable
```

CanNavigatePrevTable() as Boolean

Returns whether or not the report can navigate to the previous table. Returns false if there are no other tables.

Example

```
MsgBox AWXRep321.CanNavigatePrevTable
```

NavigateNextTable()

Navigates to the next available table. Does nothing if no other tables exist, or this is the last table already.

Example

```
MsgBox AWXRep321.NavigateNextTable
```

NavigatePrevTable()

Navigates to the previous table. Does nothing if you are already on the first table.

Example

```
MsgBox AWXRep321.NavigatePrevTable
```

PrintReport(short iReportIndex, short nCopies, long lStartPage, long lEndPage) as Short

Prints the report to the default printer from page X to page Y.

Example

```
ret = AWXRep321.PrintReport(1, 1, 1, 5)
```

PrintReportToDevice(short iReportIndex, BSTR strDriverName, BSTR strDeviceName,BSTR strPortName, short nCopies, long lStartPage, long lEndPage) as Short

Prints the report to the specified device from page X to page Y.

GetHighlightedRange() as VARIANT

Returns a pair of coordinates corresponding to the upper left selected grid and the lower right selected grid.

Example

```
Dim sel As Variant
sel = AWXRep321.GetHighlightedRange
For Each coordinate In sel
      MsgBox coordinate
Next coordinate
```

ScrollUp()

ScrollDwn()

ScrollPgUp()

ScrollPgDwn()

ScrollLeft()

ScrollRight()

ScrollPgLeft()

ScrollPgRight()

SelectGridColumn(short iIndex)

Selects the specified column in the grid.

Example

```
AWXRep321.SelectGridColumn(2)
```

GetSelectedGridColumns() as VARIANT

Returns a list of the currently selected grid columns. If no columns are selected, returns “Nothing”.

Example

```
Dim n As Variant
n = AWXRep321.GetSelectedGridColumns
```

GetAllColumns() as VARIANT

Returns a list of all columns available to the report.

Example

```
Dim cols As Variant
cols = AWXRep321.GetAllColumns
For Each col In cols
      MsgBox col
```

```
Next col
```

GetAutoPrintSupport() as IOAutoPrintSupport

Returns an OAutoPrintSupport object to be used for configuring auto printing. See the OAutoPrintSupport documentation below for its properties.

Example

```
Dim aps As OAutoPrintSupport
Set aps = AWXRep321.GetAutoPrintSupport
```

GetCRTemplate(short iReportIndex) as IOCRTemplate

Returns a Crystal Reports template object. See the OCRTemplate documentation below for more information on its properties and methods.

Example

```
Dim cr As OCRTemplate
Set cr = AWXRep321.GetCRTemplate(1)
```

ReplaceHost(BSTR OldHostName, BSTR NewHostName) as Long

Works over both data source tags and path name attributes and replaces node name **substring** within URL path only. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces host name in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceHost("Host1", "Host2")
'if Status <> 0 then there was no replacement performed or
an error occurs
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

ReplaceFilePath(BSTR OldSubstring, BSTR NewSubstring) as Long

Works over path name parameters of pick dynamics. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces file path in whole display

Dim Status As Long
```

```
Status = ThisDisplay.ReplaceFilePath("C:\Windows\Temp",
"D:\Temp")
'if Status <> 0 then there was no replacement performed or
an error occurs
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

ReplaceHostEx(BSTR OldHostNameSubstring, BSTR NewHostNameSubstring, BOOL MatchCase, BOOL MatchWholeWord) as Long

Works over both data source tags and path name attributes and replaces node name **substring** within URL path only, and supports case-sensitivity, wildcard strings and MatchWholeWord flag. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces host name in whole display, regarding "case" and
"whole words" options

Dim Status As Long
Status = ThisDisplay.ReplaceHostEx("Host1", "Host2", True,
True)
'if Status <> 0 then there was no replacement performed or
an error occurs
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

RefreshAsynch

Performs the same operation as **Refresh** method, but asynchronously: **RefreshAsynch** method returns immediately, without waiting for operation completion.

Example

```
ThisDocument.AWXRep321.RefreshAsynch
```

NavigatePrevTableAsynch

Description

Performs the same operation as **NavigatePrevTablemethod**, but asynchronously: **NavigatePrevTableAsynch** method returns immediately, without waiting for operation completion.

Example

```
ThisDocument.AWXRep321.NavigatePrevTableAsynch
```

NavigateNextTableAsynch

Description

Performs the same operation as **NavigateNextTablemethod**, but asynchronously: **NavigateNextTableAsynch** method returns immediately, without waiting for operation completion.

Example

```
ThisDocument.AWXRep321.NavigateNextTableAsynch
```

SwitchColumnsAsynch(iIndex1 As integer, iIndex2 As integer)

Performs the same operation as **SwitchColumns**, but asynchronously: **SwitchColumnsAsynch** method returns immediately, without waiting for operation completion.

Example

```
ThisDocument.AWXRep321.SwitchColumnsAsynch(1,3)
```

DeleteColumnAsynch (iIndex As integer)

Performs the same operation as **DeleteColumn**, but asynchronously: **DeleteColumnAsynch** method returns immediately, without waiting for operation completion.

Example

```
ThisDocument.AWXRep321.DeleteColumnAsynch(2)
```

UpdateColumnsInfoAsynch

Performs the same operation as **UpdateColumnsInfo**, but asynchronously: **UpdateColumnsInfoAsynch** method returns immediately, without waiting for operation completion.

Example
```
ThisDocument.AWXRep321.UpdateColumnsInfoAsynch
```

UpdateFiltersInfoAsynch

Performs the same operation as **UpdateFiltersInfo**, but asynchronously: **UpdateFiltersInfoAsynch** method returns immediately, without waiting for operation completion.

Example
```
ThisDocument.AWXRep321.UpdateFiltersInfoAsynch
```

DeleteFilterAsynch (iIndex As integer)

Performs the same operation as **DeleteFilter**, but asynchronously: **DeleteFilterAsynch** method returns immediately, without waiting for operation completion.

Example
```
ThisDocument.AWXRep321.DeleteFilterAsynch(2)
```

ClearGridSelection

Unselects all selected grid cells

Example

ThisDocument.AWXRep321.ClearGridSelection

Available Control Events

OnUpdate()

OnChartSeriesSelect(BSTR strSelectedItem)

OnGridSelectionChanged(boolean IsDragging)

OnGridLButtonDblClk(short iColIndex, long iRowIndex)

OnGridLButtonClicked(short iColIndex, long iRowIndex)

OnGridRButtonDblClk(short iColIndex, long iRowIndex)

OnGridRButtonClicked(short iColIndex, long iRowIndex)

OnGridMButtonDblClk(short iColIndex, long iRowIndex)

OnGridMButtonClicked(short iColIndex, long iRowIndex)

OnChartDblClick()

OnChartMouseUp(short Button, short Shift)

OnChartMouseDown(short Button, short Shift)

OnChartClicked()

Available Automation Objects

OGridColumnWrapper,

OGridSummaryColumnWrapper object properties:

HeaderTitle

Type: BSTR

Description

Sets/gets the value of the column header field.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
column.HeaderTitle = "New Column Header"
ThisDisplay.AWXRep321.UpdateColumnsInfo
```

The above example will change the header of column 1 in the Alarm Report ActiveX.

ColumnName

Type: BSTR

Description

Sets/gets the value of the column name field.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
Msgbox column.ColumnName
```

The above example will display the column name for column 1.

Width

Type: Short

Description

Sets/gets the width of the column

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
column.Width = 100
```

The above example will set the width of column 1 equal to 100.

DataType

Type: Short

Description

Sets/gets the data type of the column

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
Msgbox column.DataType
```

The above example will display the data type of column 1.

ModifyFont

Type: Boolean

Description

Enables/Disables the modified cell font.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
column.ModifyFont = False
```

The above example will disable the modified font for column 1.

ModifyHeadFont

Type: Boolean

Description

Enables/Disables the modified header font.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
column.ModifyHeadFont = False
```

The above example will disable modified the header font for column 1.

ModifyTextColor

Type: Boolean

Description

Enables/Disables the modified text color.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
column.ModifyTextColor = True
column.TextColor = RGB(0, 255, 0)
```

The above example will change the text color for column 1 to green.

TextColor

Type: OLE_COLOR

Description

Sets/gets the color of the column text.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
column.ModifyTextColor = True
column.TextColor = RGB(0, 255, 0)
```

The above example will change the text color for column 1 to green.

ModifyBackColor

Type: Boolean

Description

Enables/Disables the modified background color.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
column.ModifyBackColor = True
column.BackColor = RGB(0, 255, 0)
```

The above example will change the background color for column 1 to green.

BackColor

Type: OLE_COLOR

Description

Sets/gets the background color of the column.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
column.ModifyBackColor = True
column.BackColor = RGB(0, 255, 0)
```

The above example will change the background color for column 1 to green.

ModifyCell

Type: Boolean

Description

Enables/Disables changes to the cell justification and wrap text settings.

Example

```
    Dim column As OGridColumnWrapper
    Set column = ThisDisplay.AWXRep321.GetColumn(1)
    column.ModifyCell = True
    column.CellJustify = 0
The above example will change the column to left
justified.
```

CellWrapText

Type: Boolean

Description

Enables/Disables column text wrapping.

Example

```
    Dim column As OGridColumnWrapper
    Set column = ThisDisplay.AWXRep321.GetColumn(1)
    column.ModifyCell = True
    column.CellWrap = True
```

The above example will enable cell wrapping.

CellJustify

Type: Short

Description

Sets/Gets the column justification

Example

```
    Dim column As OGridColumnWrapper
    Set column = ThisDisplay.AWXRep321.GetColumn(1)
    column.ModifyCell = True
    column.CellJustify = 0
```

The above example will change the column text to left justified.

HeadJustify

Type: Short

Description

Sets/Gets the column justification

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
column.HeadJustify = 0
```

The above example will change the column header to left justified.

CellFont

Type: IfontDisp*

Description

Sets/Gets the cell font

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
Dim fnt As IFontDisp
Set fnt = column.CellFont
fnt.Bold = True
fnt.Italic = True
Set column.CellFont = fnt
```

The above example will turn on bold and italic for column 1.

HeadFont

Type: IfontDisp*

Description

Sets/Gets the header font

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.AWXRep321.GetColumn(1)
Dim fnt As IFontDisp
Set fnt = column.HeadFont
fnt.Bold = True
fnt.Italic = True
Set column.HeadFont = fnt
```

The above example will turn on bold and italic for column 1.

OFilterWrapper properties:

Active

Type: Boolean

Description

Activates/Deactivates the indicated filter

Example

```
Dim filter As ofilterwrapper
```

```
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
filter.Active = True
ThisDisplay.AWXRep321.UpdateFiltersInfo
```

The above example will activate filter number 1.

FilterName

Type: BSTR

Description

Sets/Gets the filter name

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
filter.FilterName = "New Filter"
ThisDisplay.AWXRep321.UpdateFiltersInfo
```

The above example will rename filter number 1 to the name “New Filter”.

ColumnName

Type: BSTR

Description

Sets/Gets the name of the column to filter

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
filter.ColumnName = "[Tag]"
ThisDisplay.AWXRep321.UpdateFiltersInfo
```

The above example will change the filter to filter on column “[Tag]”.

DataType

Type: Short

Description

Gets the data type of the column selected in the filter.

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
Msgbox filter.DataType
```

The above example will display the data type of the column selected in the filter.

FilterType

Type: Short

Description

Sets/Gets the type of the filter. Can be one of the following:

0 – Inclusive Filter

1 – Exclusive Filter

2 – Custom Filter

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
filter.FilterType = 0
ThisDisplay.AWXRep321.UpdateFiltersInfo
```

The above example will set the filter type as an Inclusive Filter.

IncludedValues

Type: VARIANT

Description

Sets/Gets the column values included by the filter.

Example

```
Dim myfilter As ofilterwrapper
Set myfilter = ThisDisplay.AWXRep321.GetFilter(1)
Dim included As String
Dim varstring
Dim myvar As Variant
included = ""
myvar = myfilter.IncludedValues
For Each varstring In myvar
    included = included & CStr(varstring) & "; "
Next varstring
Msgbox included
```

The above example will display a list of the included values separated by semicolons.

ExcludedValues

Type: VARIANT

Description

Sets/Gets the column values excluded by the filter.

Example

```
Dim myfilter As ofilterwrapper
Set myfilter = ThisDisplay.AWXRep321.GetFilter(1)
Dim excluded As String
Dim varstring
Dim myvar As Variant
```

```
excluded = ""
myvar = myfilter.ExcludedValues
For Each varstring In myvar
    excluded = excluded & CStr(varstring) & "; "
Next varstring
Msgbox excluded
```

The above example will display a list of the included values separated by semicolons.

TimeMode

Type: Short

Description

Sets/Gets the date/time type for inclusive or exclusive filters. Can be one of the following:

0 – Today

1 – Single Day

2 – Date Range

3 – Last

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
Filter.TimeMode = 2
ThisDisplay.AWXRep321.UpdateFiltersInfo
```

The above example will set date/time to “single day”.

Days

Type: Short

Description

Sets/Gets the last x Days value for Date/Time filters

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
Filter.Days = 1
```

The above example will set the filter to show the last 1 day.

Hours

Type: Short

Description

Sets/Gets the last x Hours value for Date/Time filters

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
Filter.Hours = 1
```

The above example will set the filter to show the last 1 hour.

Minutes

Type: Short

Description

Sets/Gets the last x Minutes value for Date/Time filters

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
Filter.Minutes = 1
```

The above example will set the filter to show the last 1 minute.

DateTimeFrom

Type: Variant

Description

Sets/Gets the Time From setting for filters that are set for Date Range. This setting is also used to determine which day is selected for Single Day filters.

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
Filter.DateTimeFrom = "7/27/2003 12:00:00 AM"
Filter.DateTimeTo = "7/28/2003 12:00:00 AM"
```

The above example will set the filter to show any message occurring on 7/27/2003.

DateTimeTo

Type: Variant

Description

Sets/Gets the Time To setting for filters that are set for Date Range.

Example

```
Dim filter As ofilterwrapper
Set filter = ThisDisplay.AWXRep321.GetFilter(1)
Filter.DateTimeFrom = "7/27/2003 12:00:00 AM"
Filter.DateTimeTo = "7/28/2003 12:00:00 AM"
```

The above example will set the filter to show any message occurring on 7/27/2003.

CustomSQL

Type: BSTR

Description

Sets/Gets the custom SQL string that is used when the Filter Type is a Custom Filter.

Example

```
    Dim filter As ofilterwrapper
    Set filter = ThisDisplay.AWXRep321.GetFilter(1)
    Filter.CustomSQL = "[Tag]
IN('Humidity','Pump1','Tank1')"
```

The above example will set the custom SQL to select messages with where the Tag field is 'Humidity', 'Pump1', or 'Tank1'.

OFilterWrapper methods:

GetSQL() as BSTR

Type: BSTR

Description

Returns the custom SQL string that is used when the Filter Type is a Custom Filter. The returned string is enclosed in parenthesis.

Example

```
    Dim filter As ofilterwrapper
    Set filter = ThisDisplay.AWXRep321.GetFilter(1)
    MsgBox filter.CustomSQL
```

The above example will return the Custom SQL string.

IsValid() as Boolean

Type: BSTR

Description

Returns whether or not the filter is valid.

Example

```
    Dim filter As ofilterwrapper
    Set filter = ThisDisplay.AWXRep321.GetFilter(1)
    If filter.IsValid Then
        MsgBox "This filter is valid."
    Else
        MsgBox "This filter is invalid."
    End If
```

The above example will say let you know if the filter is valid.

OSortEntryWrapper properties:

ColumnName

Type: BSTR

Description

Sets/Gets the column name to sort on.

Example

```
Dim sort As OSortEntryWrapper
Set sort = ThisDisplay.AWXRep321.GetSortEntry(1)
sort.ColumnName = "[Tag]"
ThisDisplay.AWXRep321.UpdateSortInfo
```

The above example sets the sort column as [Tag].

SortDir

Type: Short

Description

Sets/Gets sort direction. Possible values are:

0 – Sort Ascending

1 – Sort Descending

2 – Do not sort on this field

Example

```
Dim sort As OSortEntryWrapper
Set sort = ThisDisplay.AWXRep321.GetSortEntry(1)
Sort.sortdir = 1
ThisDisplay.AWXRep321.UpdateSortInfo
```

The above example sets the sort direction as ascending.

ResortMode

Type: Short

Description

Sets/Gets the re-sort settings. Possible values are:

0 – Do not re-sort after translating

1 – Re-sort after translating

Example

```
Dim sort As OSortEntryWrapper
Set sort = ThisDisplay.AWXRep321.GetSortEntry(1)
Sort.ResortMode = 1
ThisDisplay.AWXRep321.UpdateSortInfo
```

The above example sets the sort entry to re-sort after translating

OAutoPrintSupport properties:

EnableAutoPrint

Type: Boolean

Description

Enables/Disables auto printing of Alarm Report

Example

```
    Dim autoprint As OAutoPrintSupport
    Set autoprint =
ThisDisplay.AWXRep321.GetAutoPrintSupport()
    autoprint.EnableAutoPrint = True
```

The above example enables auto print support.

AutoPrintMode

Type: Short

Description

Gets/Sets the printing mode. Possible values are:

0 – Default printer

1 – Selected Devices

Example

```
    Dim autoprint As OAutoPrintSupport
    Set autoprint =
ThisDisplay.AWXRep321.GetAutoPrintSupport()
    autoprint.AutoPrintMode = 0
```

The above example selects default printer as the printing mode.

Orientation

Type: Short

Description

Gets/Sets the printer orientation. Possible values are:

0 – Portrait

1 – Landscape

Example

```
    Dim autoprint As OAutoPrintSupport
    Set autoprint =
ThisDisplay.AWXRep321.GetAutoPrintSupport()
    autoprint.Orientation = 0
```

The above example selects portrait as the orientation.

NumberOfCopies

Type: Short

Description

Gets/Sets the number of copies to be printed.

Example

```
    Dim autoprint As OAutoPrintSupport
    Set autoprint =
ThisDisplay.AWXRep321.GetAutoPrintSupport()
    Autoprint.NumberofCopies = 5
```

The above example sets the number of copies to 5.

PrintContent

Type: Short

Description

Gets/Sets content of the pages to be printed. Possible values are:

0 - None

1 – Crystal Report – Design #5

2 – Crystal Report – Design #4

4 – Crystal Report – Design #3

8 – Detailed Alarm Report

16 – Condensed Alarm Report

32 – Current Grid / Chart View

* To include multiple choices in auto-printing just add the values together. For example, 63 = all report views.

Example

```
    Dim autoprint As OAutoPrintSupport
    Set autoprint =
ThisDisplay.AWXRep321.GetAutoPrintSupport()
    autoprint.PrintContent = 63
```

The above example sets all pages to be printed.

SpecialPrinter1

Type: IOSpecialPrinter*

Description

Gets/Sets settings for an additional printer.

Example

```
    Dim autoprint As OAutoPrintSupport
    Set autoprint =
ThisDisplay.AWXRep321.GetAutoPrintSupport()
    Dim specprint As OSpecialPrinter
    Set specprint = autoprint.SpecialPrinter1()
    specprint.Orientation = 0
```

The above example sets the orientation of the additional printer as portrait.

IOSpecialPrinter* SpecialPrinter2

Type: IOSpecialPrinter*

Description

Gets/Sets settings for an additional printer.

Example

See Example for SpecialPrinter1

IOSpecialPrinter* SpecialPrinter3

Type: IOSpecialPrinter*

Description

Gets/Sets settings for an additional printer.

Example

See Example for SpecialPrinter1

IOSpecialPrinter* SpecialPrinter4

Type: IOSpecialPrinter*

Description

Gets/Sets settings for an additional printer.

Example

See Example for SpecialPrinter1

IOSpecialPrinter* SpecialPrinter5

Type: IOSpecialPrinter*

Description

Gets/Sets settings for an additional printer.

Example

See Example for SpecialPrinter1

OSpecialPrinter properties:

DriverName
Type: BSTR

Description
Gets/Sets the driver name for the printer.

Example

```
Dim specprint As OSpecialPrinter
Msgbox specprint.DriverName
```

This example displays the name of the driver.

DeviceName
Type: BSTR

Description
Gets/Sets the device name for the printer.

Example

```
Dim specprint As OSpecialPrinter
Msgbox specprint.DeviceName
```

This example displays the name of the device.

PortName
Type: BSTR

Description
Gets/Sets the port name.

Example

```
Dim specprint As OSpecialPrinter
Msgbox Specprint.PortName
```

This example displays the name of the port.

NumberOfCopies
Type: short

Description

Gets/Sets the number of copies to print.

Example

```
Dim specprint As OSpecialPrinter
specprint.NumberOfCopies = 5
```

Orientation
Type: short

Description
Gets/Sets the orientation for printing

Example

```
Dim specprint As OSpecialPrinter
specprint.Orientation = 0
```

This example sets the orientation as portrait.

OCRLine properties:

long Style

long PenWidth

OLE_COLOR Color

OCRText properties:

BSTR Text

IFontDisp* Font;

short FontPointSize

OLE_COLOR ForeColor

OLE_COLOR BackColor

OLE_COLOR BorderColor

short Justify

short Left

short Width

short Top

short Height

bool DropShadow

long LeftBorder

long TopBorder

long RightBorder

long BottomBorder

OCRField properties:

BSTR Field

BSTR HeaderText

short Width

OCRTemplate properties:

OLE_COLOR DataColor

IFontDisp* HeaderFont

short HeaderFontPointSize

IFontDisp* DataFont

short DataFontPointSize

OLE_COLOR HeaderColor

IOCRText* MainTitle

IOCRText* SubTitle

IOCRLine* HeaderDivider

IOCRLine* DataRowDivider

short XtraHeaderSpace

short XtraDataSpace

BSTR TemplateName

float BottomMargin

float TopMargin

float LeftMargin

float RightMargin

short PaperType

short ReportingMode

boolean HasUpperLimit

long UpperLimit

boolean AttachCurrTableName

short Orientation

short SortingMode

OCRTemplate methods:

GetFieldsCount() as Short

AddField(BSTR NewFieldName, BSTR NewFieldHeaderText, short NewFieldWidth) as IOCRField

DeleteField(short FieldIndex)

SwitchFields(short FieldIndex1, short FieldIndex2)

GetFieldObject(short FieldIndex) as IOCRField

GetFiltersNumber() as Short

GetSortEntriesNumber() as Short

GetFilter(short iIndex) as IOFilterWrapper

AddFilter(BSTR newCol) as IOFilterWrapper

DeleteFilter(short iIndex)

GetSortEntry(short iIndex) as IOSortEntryWrapper

AddSortEntry(BSTR newCol) as IOSortEntryWrapper

DeleteSortEntry(short iIndex)

SwitchSortEntries(short iIndex1, short iIndex2)

AlarmWorX32 Viewer ActiveX OLE Automation Reference

This reference describes the OLE Automation features available in the AlarmWorX32 Viewer ActiveX.

Automation Interfaces

The AlarmWorX32 Viewer ActiveX provides a COM interface that allows automation interfaces run from within the ActiveX container to manipulate the Viewer as it is running. The interface is available to all programming languages that support COM including Microsoft Visual Basic (VB), Visual Basic for Applications (VBA) and Microsoft Visual C^{++}.

To access the automation interface from VB and VBA, the AlarmWorX32 Viewer must be made available by choosing **Project - Components** from the main menu in the VB or VBA development environment and selecting **AWXVIEW32 ActiveX** in the list of available components.

Available Control Properties

boolean Check

Description

Enable/Disable URL file support.

Remarks

If check is True URL file support is enabled. FileEnabled must also be True. URL file support is only available for loading.

Example

```
AWXVIEW32OCX1.Check = True
```

boolean FileEnabled

Description

Enable/Disable file storage

Remarks

If FileEnabled is True, file support is enabled. File support conisists of loading and saving configuration information to a file other than the container document. It is recommended that the LoadSaveFile Method be used instead.

Example

```
AWXVIEW32OCX1.FileEnabled = True
```

BSTR Caption

Description

String used for Title

Remarks

This property has no function at this time.

BSTR Filename

Description

Name of the file to load/save

Remarks

If FileEnabled is True, file support is enabled. The Filename is the name of the file and the path of the file to load and save configuration information to. File names must have an .awv extension. It is recommended that the LoadSaveFile Method be used instead.

Example

Sets the file name

```
AWXVIEW32OCX1.Filename = "C:\test.awv"
```

Gets the file name

```
Dim Name as String
Name = AWXVIEW32OCX1.Filename
```

BSTR UrlPath

Description

Name of the URL path and file to load from

Remarks

The UrlPath is the name of the file and the path of the file to load configuration information from. File names must have an .awv extension. Please note that configuration information cannot be saved to a URL file.

Example

Sets the URL file name

```
AWXVIEW32OCX1.UrlPath = "http:\12.13.150.2\test.awv"
```

Gets the URL file name

```
Dim Name as String
Name = AWXVIEW32OCX1.UrlPath
```

boolean AutoSize

Description

Enable/Disable AutoSize as default sizing method.

Remarks

AutoSize determines the base size of a column based on the size of the header text of that column.

Example

```
AWXVIEW32OCX1.AutoSize = True
```

boolean DispRow

Description

Enable/Disable displaying row header

Remarks

Displaying the row header enables an operator to change the row width size during runtime. This property can be used to with the NumRowCheck property to provide a numerical value for each row.

Example

```
AWXVIEW32OCX1.DispRow = True
```

boolean ManSize

Description

Enable/Disable Manual Size as default size

Remarks

Enabling this property will prevent the viewer from altering the sizes of any columns except for the last column. Use this property if you wish to manually set base column sizes.

Example

```
AWXVIEW32OCX1.AutoSize = False
```

boolean NumRowCheck

Description

Enable/Disable numbering row header. Only enabled when DispRow is Enabled.

Remarks

Use this property in conjunction with the DispRow property to display a number for each alarm line.

Example

Example displays the number of each alarm line

```
AWXVIEW32OCX1.DispRow = True
AWXVIEW32OCX1.NumRowCheck = True
```

boolean Scale

Description

N/A

Remarks

This property has no function at this time.

boolean Title

Description

N/A

Remarks

This property has no function at this time.

boolean WrapText

Description

Enable/Disable text wrapping. Only visible when row size is greater than 1.

Remarks

Enabling this property causes the all column text to wrap to the next line. The wrap effect is contained within each individual column cell. Please note that this effect is only visible if the RowSize property is greater than 1 and the text information is larger than one column length.

Example

This example sets the Column height to 2 and enables text wrapping.

```
AWXVIEW32OCX1.WrapText = True
AWXVIEW32OCX1.RowSize = 2
```

long MaxLength

Description

Sets default cell length.

Remarks

This property will set all columns in the viewer to the sizes specified (excluding the last column). Use this property if you wish to set all base column sizes to the same size.

Example

```
AWXVIEW32OCX1.MaxLength = 10
```

JUSTIFY BaseJustify

Description

Sets default justification.

Remarks

This property sets the base justification for each column. It makes use of the enumerated values JustifyCenter, JustifyLeft, JustifyRight. The second half of the enumerated value name denotes they type of justification. For example JustifyCenter would center the text for each column.

Example

```
AWXVIEW32OCX1.BaseJustify = JustifyCenter
```

long RowSize

Description

Sets base Row height.

Remarks

This property directly effect how much of the screen space is taken up for each row.

Example

This example sets the Column height to 2.

```
AWXVIEW32OCX1.RowSize = 2
```

OLE_COLOR BackColor

Description

Sets the default background color

Remarks

This color is used for the base background color of each row (excluding row and column headers).

RGB defines colors in the order of red, green, blue. A good trick to use when picking colors is to open a graphics package and use the color palate to determine the RGB numbers. This will allow a visual representation of the color and provide the exact corresponding RGB values.

Example

```
AWXVIEW32OCX1.BackColor = RGB(255, 255, 255)
```

OLE_COLOR HeaderColor

Description

Sets the default Header color

Remarks

RGB defines colors in the order of red, green, blue. A good trick to use when picking colors is to open a graphics package and use the color palate to determine the RGB numbers. This will allow a visual representation of the color and provide the exact corresponding RGB values.

Example

```
AWXVIEW32OCX1.HeaderColor = RGB(255, 255, 255)
```

OLE_COLOR TextColor

Description

Sets the default text color

Remarks

RGB defines colors in the order of red, green, blue. A good trick to use when picking colors is to open a graphics package and use the color palate to determine the RGB numbers. This will allow a visual representation of the color and provide the exact corresponding RGB values.

Example

```
AWXVIEW32OCX1.TextColor = RGB(255, 255, 255)
```

OLE_COLOR GridColor

Description

Sets the default background color

Remarks

This color is used for all grid space not covered by a row.

RGB defines colors in the order of red, green, blue. A good trick to use when picking colors is to open a graphics package and use the color palate to determine the RGB numbers. This will allow a visual representation of the color and provide the exact corresponding RGB values.

Example

```
AWXVIEW32OCX1.BackColor = RGB(255, 255, 255)
```

GRIDLINE HStyle

Description

Sets horizontal grid style

Remarks

Grid styles other than solid can not have widths greater than 1. Also some grid styles, other than solid, have trouble printing on some printers.

GRIDLINE is an enumerated list containing the following values: GridLineDash, GridLineDashdot, GridLineDashdotdot, GridLineDot, GridLineSolid.

Example

Example of a valid grid style and width settings

```
AWXVIEW32OCX1.HStyle = GridLineSolid
AWXVIEW32OCX1.HWidth = 2

AWXVIEW32OCX1.HStyle = GridLineDot
```

```
AWXVIEW32OCX1.HWidth = 1
```

GRIDLINE VStyle

Description

Sets vertical grid style

Remarks

Grid styles other than solid can not have widths greater than 1. Also some grid styles, other than solid, have trouble printing on some printers.

GRIDLINE is an enumerated list containing the following values: GridLineDash, GridLineDashdot, GridLineDashdotdot, GridLineDot, GridLineSolid.

Example

Example of a valid grid style and width settings

```
AWXVIEW32OCX1.VStyle = GridLineSolid
AWXVIEW32OCX1.VWidth = 2

AWXVIEW32OCX1.VStyle = GridLineDot
AWXVIEW32OCX1.VWidth = 1
```

boolean HCheck

Description

Enable/Disable horizontal grid lines

Example

```
AWXVIEW32OCX1.HCheck = True
```

boolean VCheck

Description

Enable/Disable vertical grid lines

Example

```
AWXVIEW32OCX1.VCheck = True
```

long HWidth

Description

Sets horizontal grid line width

Remarks

Grid styles other than solid cannot have widths greater than 1. Also some grid styles, other than solid, have trouble printing on some printers.

Example

Example of a valid grid style and width settings

```
AWXVIEW32OCX1.HStyle = GridLineSolid
```

```
AWXVIEW32OCX1.HWidth = 2

AWXVIEW32OCX1.HStyle = GridLineDot
AWXVIEW32OCX1.HWidth = 1
```

long VWidth

Description

Sets vertical grid line width

Remarks

Grid styles other than solid can not have widths greater than 1. Also some grid styles, other than solid, have trouble printing on some printers.

Example

Example of a valid grid style and width settings

```
AWXVIEW32OCX1.VStyle = GridLineSolid
AWXVIEW32OCX1.VWidth = 2

AWXVIEW32OCX1.VStyle = GridLineDot
AWXVIEW32OCX1.VWidth = 1
```

OLE_COLOR HorColor

Description

Sets horizontal grid line color

Remarks

RGB defines colors in the order of red, green, blue. A good trick to use when picking colors is to open a graphics package and use the color palate to determine the RGB numbers. This will allow a visual representation of the color and provide the exact corresponding RGB values.

Example

```
AWXVIEW32OCX1.HorColor = RGB(255, 255, 255)
```

OLE_COLOR VerColor

Description

Sets vertical grid line color

Remarks

RGB defines colors in the order of red, green, blue. A good trick to use when picking colors is to open a graphics package and use the color palate to determine the RGB numbers. This will allow a visual representation of the color and provide the exact corresponding RGB values.

Example

```
AWXVIEW32OCX1.VerColor = RGB(255, 255, 255)
```

boolean AutoStart

Description

Enable/Disable AutoStart feature

Remarks

AutoStart allows the viewer to enter a runtime state when the container has AmbientUserMode set to True.

Example

```
AWXVIEW32OCX1.AutoStart = True
```

boolean NoToolTips

Description

Enable/Disable NoToolTips as runtime option

Remarks

Setting this property to False will gray out the NoToolTips option on the right click runtime pop-up menu.

Example

```
AWXVIEW32OCX1.NoToolTips = False
```

boolean Help

Description

Enable/Disable Help ToolTips as runtime option

Remarks

Setting this property to False will gray out the Help option on the right click runtime pop-up menu.

Example

```
AWXVIEW32OCX1.Help = False
```

boolean Event

Description

Enable/Disable Event ToolTips as runtime option

Remarks

Setting this property to False will gray out the Event option on the right click runtime pop-up menu.

Example

```
AWXVIEW32OCX1.Event = False
```

boolean Context

Description

Enable/Disable Context ToolTips as runtime option

Remarks

Setting this property to False will gray out the Context option on the right click runtime pop-up menu.

Example

```
AWXVIEW32OCX1.Context = False
```

boolean Comment

Description

Enable/Disable Comment ToolTips as runtime option

Remarks

Setting this property to False will gray out the Comment option on the right click runtime pop-up menu.

Example

```
AWXVIEW32OCX1.Comment = False
```

boolean AlarmLine

Description

Enable/Disable entire alarm line popup ToolTips as runtime option

Remarks

Setting this property to False will gray out the AlarmLine option on the right click runtime pop-up menu.

Example

```
AWXVIEW32OCX1.AlarmLine = False
```

TOOLTIP DefaultTooltip

Description

Set default ToolTip setting for runtime.

Remarks

This property sets ToolTip to use on startup. Valid ToolTip values are as follows:

- PopupAlarmLine
- PopupComment
- PopupContext
- PopupHelp
- PopupNone

Example

AWXVIEW32OCX1.Comment = False

short TooltipWidth

Description

Set tooltip width.

Remarks

Longer ToolTip widths will cause the ToolTip to have fewer rows.

Example

AWXVIEW32OCX1.TooltipWidth = 15

long CommentHeader

Description

Defines a header to associate with the Comment pop-up ToolTips

Remarks

This call should be used in conjunction with the position of the desired header. This causes the comment option to display the information in the associated column.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.ColumnNametoPosition("Column1")
AWXVIEW32OCX1.CommentHeader = Val1
```

long HelpHeader

Description

Defines a header to associate with the Help pop-up ToolTips

Remarks

This call should be used in conjunction with the position of the desired header. This causes the help option to display the information in the associated column.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.ColumnNametoPosition("Column1")
AWXVIEW32OCX1.HelpHeader = Val1
```

boolean AlarmDouble

Description

Enable/Disable left mouse double click to acknowledge alarms

Example

```
AWXVIEW32OCX1.AlarmDouble = False
```

boolean AckDialog

Description

Enable/Disable the forcing of alarm acknowledgement through ack dialog.

Remarks

Enabling this property forces all acknowledgements through the ack dialog. This differs from the AlarmDouble property in the fact that a double click with AckDailog set to true will bring up the acknowledge dialog.

Example

```
AWXVIEW32OCX1.AckDialog = False
```

boolean EnableMoveCol

Description

Enable/Disable the changing of column order during runtime

Example

```
AWXVIEW32OCX1.EnableMoveCol = True
```

boolean EnableResizeCol

Description

Enable/Disable the resizing of columns during runtime

Remarks

This feature should only be used for design and testing due to the fact that it would allow the user to completely hide a column of information.

Example

```
AWXVIEW32OCX1.EnableResizeCol = True
```

boolean EnableResizeRow

Description

Enable/Disable the resizing of rows during runtime

Remarks

This feature should only be used for design and testing due to the fact that it would allow the user to completely hide alarm lines.

Example

```
AWXVIEW32OCX1.EnableResizeRow = True
```

BSTR SelectedSource

Description

Returns the selected source string. Source name is chosen based on messages selected, alarm state, severity, and time

Remarks

This property will return the source string (tag name) that best fits the following criteria:

1) Highlighted alarm line.

2) Highest alarm state.

3) Highest severity.

4) Most recent time stamp.

Alarm state priority is defined as follows:

1) In Alarm and unacknowledged

2) In Alarm but acknowledged

3) Passed out of Alarm but unacknowledged

4) Tracking message

5) Operator message

Example

```
Dim SourceName As String
SourceName = AWXVIEW32OCX1.SelectedSource
MsgBox (SourceName)
```

BSTR PrioritySource

Description

Returns the priority source string. Source name is chosen based on messages in alarm, severity, and time.

Remarks

This property returns a source name (tag name) that best fits the following criteria:

1) In alarm.

2) Highest severity.

3) Most recent time stamp.

If no alarm messages are in alarm and unacknowledged this property will return an empty string.

Example

```
Dim SourceName As String
SourceName = AWXVIEW32OCX1.PrioritySource
MsgBox (SourceName)
```

Available Control Methods

boolean ShowColumn(COLIMNID ColumnId, boolean bShow)

Description

This method will hide and show columns. If bShow is True it will attempt to show the column. Returns a True on success.

Example

This example shows the column alarm type

```
Dim bVal as Boolean
bVal = AWXVIEW32OCX1.ShowColumn(ALARMTYPE,True)
```

boolean SetToolTip(TOOLTIP Id)

Description

This method will set the current Tooltip mode to the type associated with TOOLTIP id.

Remarks

See the DefaultTooltip property for TOOLTIP id list.

Example

```
Dim bVal as Boolean
bVal = AWXVIEW32OCX1.SetToolTip(PopupAlarmline)
```

boolean SortOn(short Count, COLIMNID Sort1, boolean Direction1, COLIMNID Sort2, boolean Direction2, COLIMNID Sort3, boolean Direction3, COLIMNID Sort4, boolean Direction4, COLIMNID Sort5, boolean Directon5)

Description

Sorts grid. Pass in columnID and direction combinations to set up sort. Count is the number of items to sort on. Pass zero for all unused parameters. The first pair can not be zero. Returns true on success. A direction of True causes accending order.

Remarks

The count parameter is used to determin how many columnID and direction pairs to use for sorting.

Example

This example causes the grid to be sorted first by alarm type and then by time. The effect is to have the earliest most recent alarms on top and the oldest operator messages on the bottom.

```
Dim bVal as Boolean
```

```
bVal = AWXVIEW32OCX1.SortOn(2, ALARMTYPE, True,
AEACTIVETIME,False , 0, 0, 0, 0, 0, 0)
```

boolean SetColumnName(BSTR Name, short Pos)

Description

This method sets the name of the column located at the possition passed in.

Remarks

Column position can change due to runtime user interaction or script. Please be aware that column position counts hidden columns when doing calculations.

Example

```
Dim bVal As Boolean
bVal = AWXVIEW32OCX1.SetColumnName("NewCol1",3)
```

short ColumnNametoPosition(BSTR Name)

Description

Pass in string name (Case sensitive) returns column position. A return of zero is a failure.

Remarks

Column position can change due to runtime user interaction or script. Please be aware that column position counts hidden columns when doing calculations.

Example

```
Dim ColPos as Integer
ColPos = AWXVIEW32OCX1.ColumnNametoPosition("Column1")
```

short ColumnNametoId(BSTR Name)

Description

Pass in string name (Case sensitive) returns column id. A Value of zero is a failure.

Remarks

The column id value corresponds to the following list:

AECHANGEMASK = 0,

AENEWSTATE = 1,

AESOURCE = 2,

AETIME = 3,

AEMESSAGE = 4,

AEEVENTTYPE = 5,

AEEVENTCATEGORY = 6,

AESEVERITY = 7,

AECONDITIONNAME = 8,

AESUBCONDITIONNAME = 9,

AEQUALITY = 10,

AEACKREQUIRED = 11,

AEACTIVETIME = 12,

AECOOKIE = 13,

AENUMEVENTATTRS = 14,

AEACTORID = 15,

ATTRIB1 = 16,

ATTRIB2 = 17,

ATTRIB3 = 18,

ATTRIB4 = 19,

ATTRIB5 = 20,

ATTRIB6 = 21,

ATTRIB7 = 22,

ATTRIB8 = 23,

ATTRIB9 = 24,

ATTRIB10 = 25,

ATTRIB11 = 26,

ATTRIB12 = 27,

ATTRIB13 = 28,

ATTRIB14 = 29,

ATTRIB15 = 30,

ATTRIB16 = 31,

ATTRIB17 = 32,

ATTRIB18 = 33,

ATTRIB19 = 34,

ATTRIB20 = 35,

ALARMTYPE = 36,

Example

```
Dim ColId as Integer
ColId = AWXVIEW32OCX1. ColumnNametoId("Column1")
```

BSTR IdtoColumnName(COLIMNID Id)

Description

Pass in columnID. Returns string name.

Remarks

See ColumnNametoId function for Id list.

Example

```
Dim ColName as String
ColName = AWXVIEW32OCX1.IdtoColumnName(AEACTIVETIME)
```

BSTR PositiontoColumnName(short Position)

Description

Pass in position. Returns string name.

Remarks

Column position can change due to runtime user interaction or script. Please be aware that column position counts hidden columns when doing calculations.

Example

```
Dim ColName as String
ColName = AWXVIEW32OCX1.PositiontoColumnName(2)
```

long GetNumberofAlarms()

Description

Returns current number of items in alarm.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofAlarms()
```

long GetNumberofAcked()

Description

Returns current number of items in acknowledged state.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofAcked()
```

long GetNumberofUnacked()

Description

Returns current number of items in unacknowledged state.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofUnacked()
```

long GetNumberofTracking()

Description

Returns current number of tracking items.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofTracking()
```

long GetNumberofOperator()

Description

Returns current number of operator items.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofOperator()
```

long GetTotalNumberofMessages()

Description

Returns total number of current messages.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofMessages()
```

long GetNumberofFilterAlarms()

Description

Returns current number of items in alarm that pass through the client filter.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofFilterAlarms()
```

long GetNumberofFilterAcked()

Description

Returns current number of items in acknowledged state that pass through the client filter.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofFilterAcked()
```

long GetNumberofFilterUnacked()

Description

Returns current number of items in unacknowledged state that pass through the client filter.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofFilterUnacked()
```

long GetNumberofFilterTracking()

Description

Returns current number of tracking items that pass through the client filter.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofFilterTracking()
```

long GetNumberofFilterOperator()

Description

Returns current number of operator items that pass through the client filter.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofFilterOperator()
```

long GetTotalNumberofFilterMessages()

Description

Returns total number of current messages that pass through the client filter.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.GetNumberofFilterMessages()
```

short PositiontoId(short Pos)

Description

Pass in position returns column ID.

Remarks

Column position can change due to runtime user interaction or script. Please be aware that column position counts hidden columns when doing calculations.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.PositiontoId(2)
```

short IdtoPosition(COLIMNID Id);

Description

Pass in column ID return position.

Remarks

Column position can change due to runtime user interaction or script. Please be aware that column position counts hidden columns when doing calculations.

Example

```
Dim Val1 As Integer
Val1 = AWXVIEW32OCX1.IdtoPosition(AEACTIVETIME)
```

boolean SizeColAtPosition(short Pos, long Size);

Description

Pass in position and new size, changes size of column at that position. Return of true means success. Size must be larger than minimum width as seen in property pages. (equal to or larger than 1)

Remarks

Column position can change due to runtime user interaction or script. Please be aware that column position counts hidden columns when doing calculations. Width values are based on an internal algorithm composed of font size and type

Example

Changes column in position 2 to a width of 15

```
Dim bVal As Boolean
bVal = AWXVIEW32OCX1.SizeColAtPosition(2,15)
```

boolean SizeColWithId(COLIMNID Id, long Size)

Description

Pass in column ID and new size, changes size of column at that position. Return of true means success. Size must be larger than minimum width as seen in property pages. (equal to or larger than 1)

Remarks

Width values are based on an internal algorithm composed of font size and type

Example

Changes the time column to a width of 15

```
Dim bVal As Boolean
bVal = AWXVIEW32OCX1.SizeColWithId(AEACTIVETIME,15)
```

boolean MovCol(short FromStartPos, short FromEndPos, short ToPos)

Description

Move columns. FromStartPos is first column selected. FromEndPos is last column selected. Moves group of columns to ToPos. Cannot move past last column. Return of True means success.

Remarks

Please be aware that column position counts hidden columns when doing calculations.

Example

This Example will move columns 3, 4, and 5 in front of column 2.

```
Dim bVal As Boolean
bVal = AWXVIEW32OCX1.MovCol(3,5,2)
```

boolean EnableColTextColorOverride(COLIMNID Id, boolean Enable)

Description

Changes text color for specified column. Enable = true enables override. Returns true on success.

Example

This Example enables the text color of the time column to be overwritten and then changes the text color.

```
Dim bVal As Boolean
bVal =
AWXVIEW32OCX1.EnableColTextColorOverride(AEACTIVETIME,
True)
Call
AWXVIEW32OCX1.SetColTextColor(AEACTIVETIME,RGB(255,255,255)
)
```

boolean EnableColBackColorOverride(COLIMNID Id, boolean Enable)

Description

Changes background color for specified column. Enable = true enables override. Returns true on success.

Example

This Example enables the background color of the time column to be overwritten and then changes the background color.

```
Dim bVal As Boolean
bVal =
AWXVIEW32OCX1.EnableColBackColorOverride(AEACTIVETIME,
True)
Call
AWXVIEW32OCX1.SetColBackColor(AEACTIVETIME,RGB(255,255,255)
)
```

void SetColTextColor(COLIMNID Id, OLE_COLOR Color)

Description

Sets text color of specified column. Automatically enables text color override.

Remarks

This function will enable the text color to be overwritten.

Example

```
Dim bVal As Boolean
Call
AWXVIEW32OCX1.SetColTextColor(AEACTIVETIME,RGB(255,255,255)
)
```

void SetColBackColor(COLIMNID Id, OLE_COLOR Color)

Description

Sets background color of specified column. Automatically enables background color override.

Remarks

This function will enable the background color to be overwritten.

Example

```
Dim bVal As Boolean
Call =
AWXVIEW32OCX1.SetColBackColor(AEACTIVETIME,RGB(255,255,255)
)
```

OLE_COLOR GetColBackColor(COLIMNID Id)

Description

Returns text color of specified column. - **Note**: this is the override color.

Remarks

This returns the column background color not the base default background color.

Example

```
Dim rcolor As OLE_COLOR
rcolor = AWXVIEW32OCX1.GetColBackColor(AEACTIVETIME)
```

OLE_COLOR GetColTextColor(COLIMNID Id)

Description

Returns background color of specified column. - **Note**: this is the override color.

Remarks

This returns the column text color not the base default text color.

Example

```
Dim rcolor As OLE_COLOR
rcolor = AWXVIEW32OCX1.GetColTextColor(AEACTIVETIME)
```

boolean EnableRowTextColor(MESSAGETYPE wType, long LowSeverity, boolean Enable)

Description

Enables/Disables text color override for associated message type and rowID. Enable = True enables override. Returns true on success.

Remarks

The messagetype values are as follows:

- Alarm = 1
- Acknowledge = 2
- Unacknowledge = 3
- Operator = 4
- Tracking = 5
- Normal = 6

Example

This Example will disable the setting of the row text color for all alarm lines in alarm for severity values starting at 400 to the next configured severity or 1000. Which ever is less.

Dim bVal As Boolean

```
bVal = AWXVIEW32OCX1.EnableRowTextColor(Alarm,400,False)
```

boolean EnableRowBackColor(MESSAGETYPE wType, long LowSeverity, boolean Enable)

Description

Enables/Disables background color override for associated message type and rowID. Enable = True enables override. Returns true on success.

Remarks

See EnableRowTextColor method for available messagetype values.

Example

This Example will disable the setting of the row background color for all alarm lines in alarm for severity values starting at 400 to the next configured severity or 1000. Which ever is less.

Dim bVal As Boolean

```
bVal = AWXVIEW32OCX1.EnableRowBackColor(Alarm,400,False)
```

void SetRowTextColor(MESSAGETYPE wType, long LowSeverity, OLE_COLOR Color)

Description

Sets text color associated with message type and rowID. Automatically enables text color override.

Remarks

See EnableRowTextColor method for available messagetype values.

Example

This Example will change the row text color for all alarm lines in alarm for severity values starting at 400 to the next configured severity or 1000. Which ever is less.

```
Call
AWXVIEW32OCX1.SetRowTextColor(Alarm,400,RGB(255,255,255))
```

void SetRowBackColor(MESSAGETYPE wType, long LowSeverity, OLE_COLOR Color)

Description

Sets background color associated with message type and rowID. Automatically enables background color override.

Remarks

See EnableRowTextColor method for available messagetype values.

Example

This Example will change the row background color for all alarm lines in alarm for severity values starting at 400 to the next configured severity or 1000. Which ever is less.

```
Call
AWXVIEW32OCX1.SetRowBackColor(Alarm,400,RGB(255,255,255))
```

OLE_COLOR GetRowTextColor(MESSAGETYPE wType, long LowSeverity)

Description

Returns the row text color of the associated message type rowID combination. **Note** - this is the override color.

Remarks

See EnableRowTextColor method for available messagetype values.

Example

This example will get the row text color value for severity values starting at 400 to the next configured severity or 1000. Which ever is less.

```
Dim rcolor As OLE_COLOR
rcolor = AWXVIEW32OCX1.GetRowTextColor(Alarm,400)
```

OLE_COLOR GetRowBackColor(MESSAGETYPE wType, long LowSeverity)

Description

Returns the row background color of the associated message type rowID combination. **Note** - this is the override color.

Remarks

See EnableRowTextColor method for available messagetype values.

Example

This example will get the row background color value for severity values starting at 400 to the next configured severity or 1000. Which ever is less.

```
Dim rcolor As OLE_COLOR
rcolor = AWXVIEW32OCX1.GetRowBackColor(Alarm,400)
```

GlobalAck(BSTR comment)

Description

Acknowledges all received alarms.

Remarks

The comment is passed to the server.

Example

```
Call AWXVIEW32OCX1.GlobalAck("down for maintenance")
```

AckSelectedRows(BSTR comment)

Description

Acknowledges rows selected by the operator.

Remarks

The comment is passed to the server.

Example

```
Call AWXVIEW32OCX1.AckSelectedRows("down for maintenance")
```

void AckDisplayed(BSTR comment)

Description

Acknowledges all displayed alarms

Remarks

This function will acknowledge all alarm messages visible on the screen. The comment is passed to the server.

Example

```
Call AWXVIEW32OCX1.AckDisplayed("down for maintenance")
```

void AckFiltered(BSTR comment)

Description

Acknowledges all alarms that pass through the client filter. This differs from the global acknowledge by the fact that the global acknowledge will also acknowledge requested alarms that the client filter has filtered out.

Remarks

The comment is passed to the server.

Example

```
Call AWXVIEW32OCX1.AckFiltered("down for maintenance")
```

void SetActorID(BSTR Operator, BSTR Station)

Description

Creates string of defining user who acknowledged an alarm. Syntax is as follows user: Operator Node: Station

Remarks

Once ActorID is called the ocx will not get the ActorID from the security system unless resetActorId is called.

Example

```
Call AWXVIEW32OCX1.SetActorID("NewOperator","NewStation")
```

void SetSecurity(long Security)

Description

Allows the AlarmWorX viewer security to be set through an automation interface.

Remarks

Once enabled, Iconics security server will not be used for the life of the ocx. The security map is as follows:

m_bSecAckPoint	0x0001
m_bSecGlobalAck	0x0002
m_bSecPageAck	0x0004
m_bSecScreenAck	0x0008
m_bSecAreaAck	0x0010
m_bSecVBEvent	0x0020
m_bSecSort	0x0040
m_bSecFilter	0x0080
m_bSecDetails	0x0100
m_bSecProperty	0x0200
m_bSecApplication	0x0400
m_bSecDisplay	0x0800
m_bSecTooltip	0x1000
m_bSecColumn	0x2000
m_bSecLanguage	0x4000
m_bSecChFilter	0x8000

A 1 enables the feature and a 0 disables the feature. By default all items are disabled.

Example

This Example enables all options except AckPoint (point acknowledge)

```
Dim Security As Integer
Security = 65534 '65534  = 0xfffe
Call AWXVIEW32OCX1.SetSecurity(Security)
```

void ResetActorID()

Description

Sets the Actor ID back to the internal actor id string.

Example

```
Call AWXVIEW32OCX1.ResetActorID()
```

void FilterOff()

Description

Turns off all client filters.

Example

```
Call AWXVIEW32OCX1.ResetActorID()
```

BSTR GetActiveFilter()

Description

Returns the active client fitler name.

Remarks

This is the name of the filter not the filter expression

Example

```
Dim FilterName As String
FilterName = AWXVIEW32OCX1.GetActiveFilter()
```

BSTR GetFilter(LPCTSTR Filter)

Description

Returns the filter expression.

Remarks

Given the name of an existing filter this method will return the filter expression string. An empty string is returned if the filter does not exist.

Example

```
Dim FilterName As String
Dim FilterExpression As String
FilterExpression = AWXVIEW32OCX1.GetFilter(FilterName)
```

boolean SetFilter(LPCTSTR Filter, BSTR Expression)

Description

Creates Filter if Filter does not already exist. Sets Expression as the expression string associated with Filter. Returns a true on success.

Remarks

The best way to ensure accurate expression syntax is to create the string using the client filter dialog and then copy the string expression. This method only adds the filter and filter name to the list of possible client filters. It does not activate the filter.

Example

This example creates a filter named filter1 with an expression that only passes alarms with a source name beginning with Tag.

```
Dim FilterName As String
Dim FilterExpression As String

FilterName = "Filter1"
'actual expression  x=  like({{Source}},$"Tag*"$,0)
FilterExpression = "like({{Source}},$"Tag*"$,0)"
FilterExpression =
AWXVIEW32OCX1.SetFilter(FilterName,FilterExpression)
```

boolean EnableFilter(LPCTSTR Filter, boolean State)

Description

Turns Filter on if state is True or off if state is False. Returns true on success.

Remarks

This method will turn filters on and off. When a filter is turned on, the previous filter will be turned off. It is only possible to have one filter running at any given time.

Example

This example turns the filter named filter1 on.

```
Dim FilterName As String
Dim bVal as Boolean
FilterName = "Filter1"
bVal = AWXVIEW32OCX1.EnableFilter(FilterName,True)
```

boolean AlarmFilter(BSTR Filter, long StartSeverity, long EndSeverity, long AlarmType, BSTR AttributeField1, BSTR Attribute1, BSTR AttributeField2, BSTR Attribute2)

Description

Sets Filter with the following syntax:

({{Severity}} >= StartSeverity)&&

({{Severity}} <= EndSeverity)&&

```
( {{AlarmType}} == AlarmType )&&
( like({{Attribute1}},$"AttributeField1"$,0) )&&
( like({{Attribute2}},$"AttributeField2"$,0) )
```

Remarks

AlarmType decodes as follows:

- ALARM 1
- ACK 2
- UNACK 3
- OPER 4
- TRACK 5
- NORM 6

An AlarmType of 7 or greater removes alarm type from the filter.

The attribute field compare the string AttributeField with the string found in the column of the associated attribute. Comparison is done using the like function. The like function behaves identically to the VB like operator.

Attibute1 is removed from the filter if AttributeField1 is the string *

Attibute2 is removed from the filter if AttributeField2 is the string *

It is recommended that the SetFilter and EnableFilter methods are used instead of this method.

Example

```
Dim bVal as Boolean
bVal = AWXVIEW32OCX1.AlarmFilter("Filter1",0,1000, Alarm,
"*","extracol1","*","extracol2")
```

void EnterAnimate()

Description

Causes the ActiveX to enter runtime mode.

Example

This example causes the viewer to exit runtime, load file test.awv, and enter runtime.

```
Dim FileName As String
Dim bVal as Boolean

FileName = "c:\test\test.awv"
Call AWXVIEW32OCX1.ExitAnimate()
```

```
BVal = AWXVIEW32OCX1.LoadSaveFile(FileName,1)
Call AWXVIEW32OCX1.EnterAnimate()
```

void ExitAnimate()

Description

Causes the ActiveX to exit runtime mode and enter design mode.

Example

This example causes the viewer to exit runtime, load file test.awv, and enter runtime.

```
Dim FileName As String
Dim bVal as Boolean

FileName = "c:\test\test.awv"
Call AWXVIEW32OCX1.ExitAnimate()
BVal = AWXVIEW32OCX1.LoadSaveFile(FileName,1)
Call AWXVIEW32OCX1.EnterAnimate()
```

boolean LoadSaveFile(BSTR FileName, short Load)

Description

Will cause the ActiveX to load or save an *.awv file. If Load is 1 then the file will be loaded. Load 0 will cause the file to be saved.

Remarks

This method will only load or save a file when the ActiveX is in design mode. The ActiveX properties Check must be False and FileEnabled must be True. This should be done through the property pages and saved as part of configuration. To set the property pages to load and save from a file, select Configuration File in the general properties tab and disable URL Path option. FileName should be the full path and filename including extension.

Example

This example causes the viewer to exit runtime, load file test.awv, and enter runtime.

```
Dim FileName As String
Dim bVal as Boolean

FileName = "c:\test\test.awv"
Call AWXVIEW32OCX1.ExitAnimate()
BVal = AWXVIEW32OCX1.LoadSaveFile(FileName,1)
Call AWXVIEW32OCX1.EnterAnimate()
```

long ControlSize(long cx, long cy)

Description

Sets the size of the ocx. Returns a 0 on success.

Remarks

Cx and cy are in windows pixel units.

Example

```
Dim size As Integer
size = AWXVIEW32OCX1.ControlSize(200,300)
```

boolean ChangeNode(BSTR OldNode, BSTR NewNode)

Description

This method changes the node name associated with the alarm subscription. This method will cause the alarm subscription to switch the network nodes it is using for data collection.

Remarks

There are possible performance hits when using this function.

Example

```
Dim bVal As Boolean
bVal = AWXVIEW32OCX1.ChangeNode("PC12","PC13")
```

void UpdateDisplay()

Description

This method causes a redraw of the internal grid information.

Remarks

This should be used at the end of a script to force the update of automation color changes.

Example

```
Dim bVal As Boolean
Call
AWXVIEW32OCX1.SetColBackColor(AEACTIVETIME,RGB(255,255,255)
)
Call
AWXVIEW32OCX1.SetColTextColor(AEACTIVETIME,RGB(255,255,255)
)
Call AWXVIEW32OCX1.UpdateDisplay()
```

BSTR SelectedColRowInfo(COLIMNID Id)

Description

This method returns a string containing the text in the column corresponding to 'Id' for the row selected in the Alarm Viewer ActiveX.

Example

```
Dim TimeInfo as string
'Get the time of the selected row
TimeInfo = AWXVIEW32OCX1.SelectColRowInfo(AETIME)
```

BSTR ColRowInfo(long Row, COLIMNID Id)

Description

This method takes the row number and column ID as input parameters. It returns the row and column information in the form of a string.

Example

```
Dim s As String
s = AWXVIEW32OCX1.ColRowInfo(1, AESOURCE)
MsgBox (s)
```

This example will pop up a message box with the source of the alarm message.

void ScrollUp();

Description

Scroll up the value of one arrow click

Example

```
AWXVIEW32OCX1.ScrollUp
```

void ScrollDwn();

Description

Scroll down the value of one arrow click

Example

```
AWXVIEW32OCX1.ScrollUp
```

void ScrollPgUp();

Description

Scroll up the value of one scroll bar click

Example

```
AWXVIEW32OCX1.ScrollPgUp
```

void ScrollPgDwn();

Description

Scroll down the value of one scroll bar click

Example

```
AWXVIEW32OCX1.ScrollPgDwn
```

void ScrollLeft();

Description

Scroll left the value of one arrow click

Example

```
AWXVIEW32OCX1.ScrollLeft
```

void ScrollRight();

Description

Scroll right the value of one arrow click

Example

```
AWXVIEW32OCX1.ScrollRight
```

void ScrollPgLeft();

Description

Scroll left the value of one scroll bar click

Example

```
AWXVIEW32OCX1.ScrollPgLeft
```

void ScrollPgRight();

Description

Scroll right the value of one scroll bar click

Example

```
AWXVIEW32OCX1.ScrollPgRight
```

long GetNumberofCol();

Description

Returns the number of columns

Example

```
Dim col As Long
```

```
col = AWXVIEW32OCX1.GetNumberofCol()
MsgBox (col)
```

This example will pop up a message box with the number of columns.

void ComputerName();

Description

Returns the local node name

Example

```
Dim s As String
s = AWXVIEW32OCX1.ComputerName()
MsgBox (s)
```

This example will pop up a message box with the computer name.

long GetColSizeAtPosition(short Pos);

Description

Pass in the column position. Returns the size of the column.

Example

```
Dim size As Long
size = AWXVIEW32OCX1.GetColSizeAtPosition(1)
msgbox (size)
```

This example will pop up a message box with the size of the first column.

long GetColSizeWithId(short Id);

Description

Pass in the column Id. Returns the size of the column.

Example

```
Dim size As Long
size = AWXVIEW32OCX1.GetColSizeWithId(AESOURCE)
MsgBox (size)
```

This example will pop up a message box with the size of the AESOURCE column.

boolean IsEnableRowTextColor(short wType, long LowSeverity);

Description

Pass in the message type and severity. Returns true if the row text color setting is true. Returns false otherwise.

Remarks

See EnableRowTextColor method for available messagetype values.

Example

This example will get the EnableRowTextColor value for severity values starting at 400 to the next configured severity or 1000. Which ever is less.

```
Dim b As Boolean
b = AWXVIEW32OCX1.IsEnableRowTextColor(Alarm,400)
```

boolean IsEnableRowBackColor(short wType, long LowSeverity);

Description

Pass in the message type and severity. Returns true if the row background color setting is true. Returns false otherwise.

Remarks

See EnableRowTextColor method for available messagetype values.

Example

This example will get the EnableRowBackColor value for severity values starting at 400 to the next configured severity or 1000. Which ever is less.

```
Dim b as Boolean
b = AWXVIEW32OCX1.IsEnableRowBackColor(Alarm,400)
```

boolean IsEnableColTextColorOverride(short Id);

Description

Pass in the column Id. Returns true if the column text color setting is true. Returns false otherwise.

Example

```
Dim b as Boolean
b = AWXVIEW32OCX1.IsEnableColTextColorOverride (AESOURCE)
```

boolean IsEnableColBackColorOverride(short Id);

Description

Pass in the column Id. Returns true if the column background color setting is true. Returns false otherwise.

Example

```
Dim b as Boolean
b = AWXVIEW32OCX1.IsEnableColBackColorOverride (AESOURCE)
```

long GetNumberofSelectedRows();

Description

Returns the number of selected rows.

Example

```
Dim l As Long
l = AWXVIEW32OCX1.GetNumberofSelectRows()
MsgBox (l)
```

This example will pop up a message box with the number of selected rows.

void SetGasParentCookie(long newCookie);

Description

For Global Aliasing System (GAS) support only

Remarks

This function is hidden and should not be used through custom scripts

Example

```
None
```

boolean SetGASThemes(BSTR strThemes);

Description

For Global Aliasing System (GAS) support only

Remarks

This function is hidden and should not be used through custom scripts

Example

None

Sub RemoveAllEvents()

Sub RemoveSelectedEvents()

Sub ShowHorzScrollbar(bShow As Boolean)

Sub ShowVertScrollbar(bShow As Boolean)

Available Control Events

OnRightMouseClick()

Description

Event posted on user right mouse click

Example

```
Private Sub AWXVIEW32OCX1_OnRightMouseClick()
'Your code here
End Sub
```

OnLeftMouseClick()

Description

Event posted on user left mouse click

Example

```
Private Sub AWXVIEW32OCX1_OnLeftMouseClick()
'Your code here
End Sub
```

OnRightMouseDBLClick()

Description

Event posted on user right double mouse click

Example

```
Private Sub AWXVIEW32OCX1_OnRightMouseDblClick()
'Your code here
```

```
End Sub
```

OnLeftMouseDBLClick()

Description

Event posted on user left double mouse click

Example

```
Private Sub AWXVIEW32OCX1_OnLeftMouseDblClick()
'Your code here
End Sub
```

OnNewAlarm()

Description

Event posted when any new message enters the viewer.

Remarks

Please Note: This event occurs when any new OPC alarm and events message is passed into the viewer. This includes client filtered messages, operator messages, tracking messages, alarm messages, and acknowledge messages. No information provided to the event that denotes message type.

Example

```
Private Sub AWXVIEW32OCX1_OnNewAlarm()
'Your code here
End Sub
```

OnPopupEvent()

Description

Event posted when the user selects the VB event menu item from the right click pop-up menu.

Example

```
Private Sub AWXVIEW32OCX1_OnPopupEvent()
'Your code here
End Sub
```

Data Mining ActiveX OLE Automation Reference

This reference describes the OLE Automation features available in the Data Mining ActiveX.

Automation Interfaces

The Data Mining ActiveX provides a COM interface that allows automation interfaces run from within the ActiveX container to manipulate the Data Mining ActiveX control as it is running. The interface is available to all programming languages that support COM, including Visual Basic (VB), Visual Basic for Applications (VBA), and Microsoft Visual C^{++}.

To access the Automation interface from VB and VBA, the Data Mining ActiveX must be made available by choosing **Project > Components** from the main menu in the VB or VBA development environment and selecting **ICONICS DBGrid** in the list of available components.

Control Properties

AutoFitColumns

Type: Boolean

Description

If true when in runtime mode, columns will automatically adjust their width to fit the contents.

AutoSelectRow

Type: Boolean

Description

If true, when the user clicks on a cell the whole row will be highlighted.

AutoUpdateInterval

Type: Short

Description

Sets/gets the number of seconds between automatic updates.

Available only for ADO connections.

Example

ThisDocument.DBMining1.AutoUpdateInterval=60

BackColor

Type: OLE_COLOR

Description

Sets/gets the background color of the column.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.DBMining1.GetColumn(1)
column.ModifyBackColor = True
column.BackColor = RGB(0, 255, 0)
```

The above example will change the background color for column 1 to green.

BlinkOnUpdate

Type: Boolean

Description

Sets/gets whether or not the Database Mining ActiveX will blink on update.

Example

ThisDocument.DBMining1.BlinkOnUpdate=True

BorderType

Type: String

Description

Sets/gets control's border appearance. Can be one of the following:

"0" - none

"1" - flat border

"2" - 3D border

Example

Sets 3D look for the control's border.

ThisDocument.DBMining1.BorderType="2"

CellHAlign

Type: Short

Description

Sets/gets the default cell alignment.

“0” – Left Alignment

“1” – Center Alignment

“2” – Right Alignment

Example

ThisDocument.DBMining1.CellHAlign="2"

ColumnHAlign

Type: Short

Description

Sets/gets the column header alignment.

“0” – Left Alignment

“1” – Center Alignment

“2” – Right Alignment

Example

ThisDocument.DBMining1. ColumnHAlign ="1"

ConnectionString

Type: String

Description

Sets/gets ADO connection string used by the Database Mining ActiveX control to access data.

Example

```
Dim strConnString as String
strConnString="Provider=Microsoft.Jet.OLEDB.4.0; _
        Data Source=c:\alarms\AWXLog32.mdb; _
        User ID=Admin; Password=SYSTEM"
ThisDocument.DBMining1. ConnectionString = strConnString
```

DataSourceType

Type: Short

Description

Sets/gets the Database Mining ActiveX connection type.

"0" - ADO

"1" – Database Mining

DBMiningTitle

Type: String

Description

Sets/gets the title shown in the window title bar of the Database Mining ActiveX control during runtime mode.

DisableResize

Type: Boolean

Description

If true, users will not be able to adjust row or column sizes.

DisableMultiSelect

Type: Boolean

Description

If true, users will not be able to select multiple rows, columns, or cells.

DisableResize

Type: Boolean

Description

Sets/gets Boolean value that controls the automatic resize of the grid columns.

DisplayGridTitle

Type: Boolean

Description

Sets/gets Boolean value that controls the visibility of the grid title bar.

Example

ThisDocument.DBMining1. DisplayGridTitle = TRUE

EnableAutoUpdate

Type: Boolean

Description

Enables/disables automatic updates for ADO connections.

Example

ThisDocument.DBMining1.EnableAutoUpdate=True

EnableWebAccess

Type: Boolean

Description

Enables/disables Database Mining ActiveX Web access for ADO connections.

Example

ThisDocument.DBMining1.EnableWebAccess=True

FileName

Type: BSTR

Description

Sets/gets the name of the Database Mining ActiveX configuration file.

Example

ThisDocument.DBMining1.FileName=”C:\DBMiningConfig.dbm”

Font

Type: StdFont

Description

Returns a reference to the control's default font.

Example

To set control's default font to Arial, 14 pt., use the following statements:

With ThisDocument.DBMining1.Font

.Name="Arial"

.Size=14

End With

ForeColor

Type: OLE_COLOR

Description

Sets/gets foreground color (cell’s font color) property of the Database Mining ActiveX.

Example

ThisDocument.DBMining1. ForeColor=RGB(0,0,0)

GridBackColor

Type: OLE_COLOR

Description

Sets/gets color value used to paint the gaps between rightmost column and right edge of the grid, last row and bottom edge of the grid, and row headers column (the 'back area' of the grid).

Example

ThisDocument.DBMining1. GridBackColor =RGB(0,127,0)

GridDefRowHeight

Type: Integer

Description

Sets/gets the default row height for data grid.

Example

ThisDocument.DBMining1. GridDefRowHeight=16

GridHeaderShade

Type: Short

Description

Sets/gets the cell effect for the grid's title bar header.

"0" = Soft Bump

"1" = Diagonal Shade

"2" = Horizontal Shade

"3" = Vertical Shade

"4" = Horizontal Bump

"5" = Vertical Bump

"6" = Metal

GridHorizLinesColor and GridVertLinesColor

Type: OLE_COLOR

Description

Sets/gets color for horizontal or vertical grid lines.

Example

ThisDocument.DBMining1. GridHorizLinesColor =RGB(127,127,127)

ThisDocument.DBMining1. GridVertLinesColor = RGB(127,127,127)

GridHorizLinesStyle and GridVertLinesStyle

Type: Integer

Description

Sets/gets style for horizontal or vertical grid lines. Can be one of the following:

0 - no line

1 - solid line

2 - dot line

3 - dash line

4 - dashdot line

5 - dashdotdot line

Example

ThisDocument.DBMining1. GridHorizLinesStyle =0

ThisDocument.DBMining1. GridVertLinesStyle = 2

GridHorizLinesWidth and GridVertLinesWidth

Type: Integer

Description

Sets/gets the widths for the horizontal or vertical grid lines when corresponding grid line style (property GridHorizLinesStyle or GridVertLinesStyle) is set to 1 - solid line. For all other line styles, these properties must be set to 1.

Example

Sets both horizontal and vertical grid lines to solid line style, 2 pixels wide.

ThisDocument.DBMining1. GridHorizLinesStyle =1

ThisDocument.DBMining1. GridVertLinesStyle = 1

ThisDocument.DBMining1. GridHorizLinesWidth =2

ThisDocument.DBMining1. GridVertLinesWidth = 2

GridShade

Type: Short

Description

Sets/gets the cell effect for the grid column‘s headers.

“0” = Soft Bump

“1” = Diagonal Shade

“2” = Horizontal Shade

“3” = Vertical Shade

“4” = Horizontal Bump

“5” = Vertical Bump

“6” = Metal

GridTitleBackColor

Type: OLE_COLOR

Description

Sets/gets the background color for the grid's title bar header.

Example

Sets title bar background color to dark blue.

ThisDocument.DBMining1. GridTitleBackColor =GRB(0,0,127)

GridTitleFont

Type: StdFont

Description

Returns a reference to the default font for the grid’s title bar header.

Example

To set control's default font to Arial, 14 pt., use the following statements:

```
With ThisDocument.DBMining1.Font
.Name="Arial"
.Size=14
End With
ThisDocument.DBMining1. GridTitleFontColor =GRB(0,0,127)
```

GridTitleHAlign

Type: Short

Description

Sets/gets the grid's title bar header alignment.

"0" – Left Alignment

"1" – Center Alignment

"2" – Right Alignment

Example

```
ThisDocument.DBMining1. GridTitleHAlign ="1"
```

GridTitleTextColor

Type: OLE_COLOR

Description

Sets/gets the text color for the grid's title bar header.

Example

Sets grid's title bar text color to yellow.

```
ThisDocument.DBMining1. GridTitleBackColor =GRB(255,255,0)
```

HeaderColor

Type: OLE_COLOR

Description

Sets/gets the background color for grid's column headers.

Example

Sets color of grid's column header to dark green.

```
ThisDocument.DBMining1. GridTitleBackColor =GRB(0,127,0)
```

HeaderFont

Type: StdFont

Description

Returns a reference to the default font for the grid's column headers.

HeaderTxtColor

Type: OLE_COLOR

Description

Sets/gets the text color for grid's column headers.

Example

Sets grid's title bar text color to yellow.

ThisDocument.DBMining1. GridTitleBackColor =RGB(255,255,0)

HideGridPopupMenu

Type: Boolean

Description

Enables/disables Database Mining ActiveX runtime popup menu.

InternalGridEdges

Type: Short

Description

Sets the amount of space between the ActiveX borders and the grid borders.

IsConnected

Type: Boolean

Description

This is read-only property, indicating that control is connected to the server.

Example

```
If  ThisDocument.DBMining1.IsConnected Then
        ``` do something here  ```
End If
```

MaxRecords

Type: Long

Description

Sets/gets the maximum number of records that the Database Mining ActiveX will download using a Web access RDS connection.

Example

ThisDocument.DBMining1.MaxRecords=1024
```

ParseHTMLTag

Type: Boolean

Description

Enables/disables the HTML parsing inside the grid's cells.

If inside a grid cell there is a string with the format:

HyperText

The Grid will show the word "HyperText" as clickable text, and when the user clicks on it an Internet Explorer window will popup opening the page "link.html".

QueryText

Type: String

Description

Sets/gets the SQL statement used with an ADO connection to populate the data grid in runtime mode.

Example

```
Dim strQry as String
strQry=ThisDocument.DBMining1.QueryText
MsgBox strQry
```

RemoteDataServer

Type: BSTR

Description

This is the Internet Information Server used with the Web access RDS connections.

Example

```
ThisDocument.DBMining1.RemoteDataServer = "http://localhost"
```

RemoteDataServerBackup

Type: BSTR

Description

This is the backup URL used with Web access RDS connections if the connection with the server specified on RemoteDataServer property fails.

Example

```
ThisDocument.DBMining1.RemoteDataServerBackup = "http://localhost"
```

ShowColsHeader and ShowRowsHeader

Type: Boolean

Description

Show/hide the rows/columns headers.

URLPathName

Type: BSTR

Description

Sets/gets the URL path for a configuration file over the Web. Must set URLPathUsed to “True” for this to take effect.

Example

ThisDocument.DBMining1.URLPathName=
”http://www.iconics.com/dbminingcfg.dbm”

URLPathUsed

Type: Boolean

Description

Enables/disables the use of a URL path.

Example

ThisDocument.DBMining1.URLPathUsed=True

UseLocalSettings;

Type: Boolean

Description

When the URL path is enabled, it allows you to use or not to use the local settings configuration file.

Example

ThisDocument.DBMining1.UseLocalSettings=True

Control Methods

AboutBox

Shows **About** dialog box.

Example

ThisDocument.DBMining1.AboutBox

Connect As Boolean

It forces the control to establish connection to database, using the information set in **ConnectionString** and **QueryText.** If control is already connected, the current connection will be closed and a new connection will be established. This method can be used in VBA scripts automatically to switch the Database Mining ActiveX between different connections.

Return Value

On success, returns TRUE; if for any reason connection cannot be established, returns FALSE.

Example

Dim strConnString as String

strConnString="Provider=Microsoft.Jet.OLEDB.4.0; _

Data Source=c:\alarms\SomeOtherDatabase.mdb; _

User ID=Admin; Password=SYSTEM"

ThisDocument.DBMining1. ConnectionString = strConnString

ThisDocument.DBMining1.QueryText="Select * from [Table]"

ThisDocument.DBMining1.Connect

ThisDocument.DBMining1.Refresh

GetColumnsNumber As Integer

Returns the number of columns currently present in the gird.

Return Value

On success, returns the number of columns; if it fails or the grid doesn't have any column, returns 0.

GetGridCellValue(long ColNum,long RowNum) as VARIANT

Takes inside the grid buffer the current cell value with column number equal to *Column* and row number equal to *RowNum*.

Return Value

On success, returns the number of columns; if it fails or the grid does not have any columns, returns 0.

GetGridCellValue(ColNum As Long, RowNum As Long) as BSTR

Returns the value of the cell that has coordinates (ColNum,RowNum).

Return Value

On success, returns the value of the cell; if the cell coordinates are not valid or if the grid is not visible on the screen, it will fail returning an empty string.

GetRowsNumber As Integer

Returns the number of rows currently present in the grid.

Return Value

On success, returns the number of rows; if it fails or the grid does not have any columns, returns 0.

GetSelectedColIndex As Integer

Returns the coordinates of the currently selected column.

Return Value

On success, returns the coordinates of the currently selected column; if it fails, returns 0.

GetSelectedRange(VARIANT* TopCrd, VARIANT* LeftCrd, VARIANT* BottomCrd, VARIANT* RightCrd)

Returns the top, left, right and bottom coordinates of the currently selected area in the grid.

Return Value

On success, returns the coordinates of the currently selected area in the grid; if it fails there is no selected area, and returns all the coordinates with a negative value.

GetSelectedRowIndex As Integer

Returns the coordinates of the currently selected row.

Return Value

On success, returns the coordinates of the currently selected row; if it fails, returns 0.

GetUnselectedCellsInRange(VARIANT* UnselectedCellList)

Returns the coordinates of any unselected cells in the range returned by GetSelectedRange. The format of the cell list will be {x1, y1, x2, y2, … }. The last element of the array will always be zero, and is not part of any coordinate.

Example:

‘This example will display the coordinates one at a time in message boxes.

```
Sub getlist(o As GwxPick)
        Dim CoordList
```

```
        Dim Coordinate
        ThisDisplay.DBMining1.GetUnselectedCellsInRange CoordList
        For Each Coordinate In CoordList
            MsgBox (Coordinate)
        Next Coordinate
End Sub
```

LoadConfigFile(BSTR newVal) as Boolean

It loads the configuration file specified in newVal.

Example

```
Dim ret As Boolean
ret = DBMining1.LoadConfigFile("C:\My Configs\cfg.dbm")
```

SaveConfigFile(BSTR newVal) as Boolean

It saves your current Database Mining ActiveX configuration to the file specified in newVal.

Example

```
Dim ret As Boolean
ret = DBMining1.SaveConfigFile("C:\My Configs\cfg.dbm")
```

LoadConfigURL(BSTR newVal) as Boolean

Loads the configuration file specified in the URL string newVal.

Example

```
Dim ret As Boolean
ret = DBMining1.LoadConfigURL("http://localhost/cfg.dbm")
```

ReplaceHost(BSTR OldHostName, BSTR NewHostName) as Long

It works over data source tags, data mining tags and path name attributes and replaces node name **substring** within URL path only. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces host name in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceHost("Host1", "Host2")
'if Status <> 0 then there was no replacement performed or an error occurs
```

```
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

ReplaceHostEx(BSTR OldHostNameSubstring, BSTR NewHostNameSubstring, BOOL MatchCase, BOOL MatchWholeWord) as Long

It works over data source tags, data mining tags and path name attributes and replaces node name **substring** within URL path only, and supports case-sensitivity, wildcard strings and MatchWholeWord flag. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces host name in whole display, regarding "case" and "whole words" options

Dim Status As Long
Status = ThisDisplay.ReplaceHostEx("Host1", "Host2", True, True)
'if Status <> 0 then there was no replacement performed or an error occurs
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

SaveConfigFile(BSTR newVal) as Boolean

It saves your current Database Mining ActiveX configuration to the file specified in newVal.

Example

```
Dim ret As Boolean
ret = DBMining1.SaveConfigFile(“C:\My Configs\cfg.dbm”)
```

ScrollDwn

Moves the Grid content of one row down performing the same action that happens when the user scrolls down the Grid’s vertical toolbar.

Example:

```
ThisDisplay.DBMining1.ScrollDwn
```

ScrollLeft

Moves the Grid content to the left performing the same action that happens when the user scrolls left the Grid's horizontal toolbar.

Example:

ThisDisplay.DBMining1.ScrollLeft

ScrollPgDwn

Moves the Grid content of one page down performing the same action that happens when the user scrolls down the Grid's vertical toolbar of one page.

Example:

ThisDisplay.DBMining1.ScrollPgDwn

ScrollPgLeft

Moves the Grid content of one page to the left performing the same action that happens when the user scrolls left the Grid's horizontal toolbar of one page.

Example:

ThisDisplay.DBMining1.ScrollPgLeft

ScrollPgRight

Moves the Grid content of one page to the right performing the same action that happens when the user scrolls right the Grid's horizontal toolbar of one page.

Example:

ThisDisplay.DBMining1.ScrollPgRight

ScrollPgUp

Moves the Grid content of one page up performing the same action that happens when the user scrolls up the Grid's vertical toolbar of one page.

Example:

ThisDisplay.DBMining1.ScrollPgUp

ScrollRight

Moves the Grid content to the right performing the same action that happens when the user scrolls right the Grid's horizontal toolbar.

Example:

ThisDisplay.DBMining1.ScrollUp

ScrollUp

Moves the Grid content of one row up performing the same action that happens when the user scrolls up the Grid's vertical toolbar.

Example:

ThisDisplay.DBMining1.ScrollUp

SetDBType(short NewDBType)

Sets the database type.

Example

DBMining1.SetDBType(1)

SortColumn(long ColID, VARIANT_BOOL bAscending)

Sorts the grid based upon a specified column in ascending or descending order.

Example:

'Sort by Column 1 ascending:

ThisDisplay.DBMining1.SortColumn 1, True

'Sort by Column 1 descending:

ThisDisplay.DBMining1.SortColumn 1, False

UnselectAllRows()

Unselects the currently selected area if present.

Example:

ThisDisplay.DBMining1.UnselectAllRows

DataWorX32 OLE Automation Reference

DataWorX32 consists of two parts – the Configurator and the Runtime part. Both expose an OLE Automation interface; however the Configurator's interface is designed for use with ProjectWorX32.

The interface of the Runtime part is described below.

The name of the GENESIS32 DwxRuntime Automation Library is **AutoDwxRuntimeLib.** It defines the following classes and interfaces:

- **DwxRuntime**
- **IPoint**
- **IRegister**
- **IRedundancyAlias**

DwxRuntime Object

This object represents the whole DataWorX32 Runtime module. It contains other DataWorX32 objects like points and registers. It can be used to create or access them.

Methods

GetPoint(strName As String, nScanRate As Long) As IPoint

Creates a point object with the specified name and scan rate and returns and interface of it. The point object can be subsequently used to read and/or write values from/to OPC servers.

The **strName** parameter should meet GENESIS32 point name convention:

i.e. [\\node\]OPCServer\TagName.

The **nScanRate** is the requested scan rate in milliseconds.

GetRegister(strName As String) As IRegister

Returns an interface of an existing DataWorX32 register. This interface can be used to access the registers value and some of its properties.

GetRedundancyAlias(strName As String) As IRedundancyAlias

Returns an interface of an existing DataWorX32 Redundancy Alias object.

ConfigurationUpdate()

Forces DataWorX32 to reload the active configuration and include changes made since the configuration was loaded last time.

Shutdown()

Forces DataWorX32 to quit immediately. Any reference to DwxRuntime object becomes invalid after calling this function. Users should not call this function – it has been primarily designed for GENESIS32 GenTray.

Property

Runtime As Boolean

This is a read-write property that tells whether DataWorX32 is in runtime mode. Writing to it starts or stops the runtime operation. When starting the runtime operation, DataWorX32 requests points on OPC servers; the data updates from the OPC servers are propagated during runtime (data bridging).

IPoint Interface

An interface to a point object can be obtained by calling the **DwxRuntime.GetPoint()** function. The point objects represent a non-persistent connection to OPC servers.

Properties

Name As String

Read-only property; name of the relevant OPC point, in the GENESIS32 point name convention.

Value As Variant

Read-write property; represents value of the relevant OPC point. Writing to this property causes an asynchronous write to the OPC server.

Method

GetValueEtc(Value, Quality As Long, Timestamp As Date, Milliseconds As Long)

Gets the current value, quality and timestamp of the relevant OPC point object.

IRegister Interface

An interface to a register existing in DataWorX32. It may represent registers of any type (i.e. aliases, switch aliases, etc.) defined in DataWorX32. This interface can be obtained by calling **DwxRuntime.GetRegister().**

Properties

Name As String

Read-only property; name of the relevant DataWorX32 register.

Value As Variant

Read-write property; represents the value of the relevant DataWorX32 register. Writing to this property changes the value of the register. If the register has defined any output, the new value may be also written into the output.

Ranges As Boolean

Read-write property telling the user whether the ranges of the DataWorX32 register are valid and should be used.

HiRange As Double

LoRange As Double

Read/write properties containing ranges of the register. Writing to these properties modifies the ranges used in runtime to clip the register value. The range changes are not persistent (do not get stored into the active configuration database.)

REGISTER_TYPE RegType

Read-only property; type of the register, one of the REGISTER_TYPE values:

RT_REGISTER = 0
RT_ALIAS = 1
RT_REDUNDANCY_ALIAS = 2
RT_REDUNDANCY_FLAG = 3
RT_SWITCH_ALIAS = 4

InputConnected As Long

Input of the register can be temporarily disconnected. For example, let's assume a register is connected to "ICONICS.Simulator\Ramp" and you want to write a value through automation into the tag. If you write the value, then the Ramp would overwrite it. So you can temporarily disconnect input (InputConnected = FALSE), write something, and then enable the connection again with InputConnected = TRUE.

Methods

GetValueEtc(Value, Quality As Long, Timestamp As Date, Milliseconds As Long)

Gets the current value, quality and timestamp of the DataWorX32 register.

Sub SetValueQuality(Value, Quality As Long)

Allows you to set both the quality and value of a register.

IRedundancyAlias Interface

An interface to a redundancy alias existing in DataWorX32. It can be obtained by calling **Application.GetRedundancyAlias().** It can be used to force a redundancy switchover.

Methods

SwitchToPrimary

Forces redundancy switchover to the node defined as primary.

SwitchToNext

Forces redundancy switchover to the backup node following the node that is currently used. If the currently used node is the last backup node, it will switch again to the primary node.

Properties

IsPrimary As Boolean

Returns TRUE if redundancy currently uses the node defined as primary; otherwise returns FALSE.

PrimaryNodeName As String

Returns the name of the node defined as primary.

CurrentNodeName As String

Returns the name of the currently used node.

GenBroker OLE Automation Reference

The **GenClientWrapper** provides an OLE automation interface that allows you to manipulate GenBroker as it is running. The interface is available to all programming languages that support COM, including Visual Basic (VB), Visual Basic for Applications (VBA), and Microsoft Visual C++.

To access the Automation interface from VB and VBA, **GenClientWrapper** must be made available by choosing the **GenClientWrapper Type Library** from the references list in the VB or VBA development environment and selecting **GenClientWrapper** from the list of available libraries.

The description of every property or method in this reference is followed by the code example for typical use of that property or method. The example code is written in Visual Basic for Applications (VBA) language.

GenClientWrapper Methods

Function ChangePassword(UserName As String, OldPassword As String, NewPassword As String) As Long

Function GetLoggedInUsers()

Description

Returns array of names of the users currently logged in to GENESIS-32

Example

```
Dim genclient As New GENCLIENTWRAPPERLib.Client

users = genclient.GetLoggedInUsers()

For Each user In users

    MsgBox user + " is logged in"

Next user
```

Function GetPointProperty(bstrPointName As String, nPropertyID As GC_POINT_PROPERTY)

Description

Returns a specified property of a specified point. See OPC Properties defined in OPC Spec for more information.

Example

```
Dim genclient As New GENCLIENTWRAPPERLib.Client

accessrights =
genclient.GetPointProperty("ICONICS.Simulator\SimulatePLC.
Sine", GC_PROPERTY_ACCESSRIGHTS)
```

```
If (accessrights < 2) Then MsgBox "not writeable"
```

Function LogIn(UserName As String, Password As String) As Long

Function LogOut(UserName As String) As Long

Function RequestDataPoint(bstrName As String, lScanRate As Long, lRequestType As Long) As DataPoint

Description

Creates a DataPoint object that represents an item on an OPC DA Server.

Example

```
Dim genclient As New GENCLIENTWRAPPERLib.Client

Dim point As GENCLIENTWRAPPERLib.DataPoint

Set point =
genclient.RequestDataPoint("ICONICS.Simulator\SimulatePLC.
Sine", 50, 0)

If (point Is Nothing) Then MsgBox "error creating point"
```

Function RequestSecurityPoint(SecType As GC_SECURITY_POINT_TYPE, bstrName As String, nFunctionID As Long) As SecurityPoint

Description

Creates a SecurityPoint object that represents an item on ICONICS Security Server. It can be used to find out if an action, OPC Tag or a file is enabled/disabled for the user currently logged in.

Example

```
Dim genclient As New GENCLIENTWRAPPERLib.Client

Dim point As GENCLIENTWRAPPERLib.SecurityPoint

Set point =
genclient.RequestSecurityPoint(GC_SECURITY_FILENAME,
"Display1.gdf", 0)

If (point Is Nothing) Then MsgBox "error creating point"
```

Function SendSimpleEvent(Source As String, Time As Date, Severity As Long, Message As String, Node As String, Comment As String) As Long

Function SendTrackingEvent(Source As String, Time As Date, Category As Long, Severity As Long, Message As String, Node As String, Comment As String, Param As String, PrevValue, NewValue) As Long

Function SendTrackingEventEx(Source As String, Time As Date, Category As Long, Severity As Long, Message As String, Node As String, Comment As String, User As String, Param As String, PrevValue, NewValue) As Long

Sub ShowStatistics()

Description

Displays runtime statistics.

Example

```
Dim genclient As New GENCLIENTWRAPPERLib.Client

Call genclient.ShowStatistics
```

Function Succeeded(gcResult As GC_RESULT) As Long

Sub GetValueEtc(Value, [Quality], [Timestamp], [TimestampMilliseconds])

Description

Member function of DataPoint object. Returns current value, quality and timestamp.

Example

```
Dim point As GENCLIENTWRAPPERLib.DataPoint

…

Call point.GetValueEtc(Value, Quality, TimeStamp)

MsgBox "Current Value: " + Str(Value)
```

Sub SyncWrite(newVal, [MillisecondsTimeout As Long = -1])

Description

Synchronously writes a value to a DataPoint

Example

```
Dim point As GENCLIENTWRAPPERLib.DataPoint

…

Call point.SyncWrite(123, 10000)
```

Function GetAccessRights(pointName As String, isReadable, isWriteable) As Boolean

Function GetRanges(pointName As String, loRange, hiRange) As Boolean

Function Read(pointName As String, [quality], [timestamp])

Sub Write(pointName As String, value)

GenClientWrapper Properties

Property LastWriteError As GC_RESULT

Property Quality As GC_DATA_POINT_QUALITY

Description

Property of DataPoint object. Contains the OPC quality of the relevant OPC Item.

Example

```
Dim point As GENCLIENTWRAPPERLib.DataPoint

Set point =
genclient.RequestDataPoint("ICONICS.Simulator\SimulatePLC.
Sine", 50, 0)


If (point.Quality = GC_QUALITY_GOOD) Then MsgBox "The
point's value is valid"
```

Property State As GC_POINT_STATE

Description

Property of both DataPoint and SecurityPoint objects. Contains the state of the object. Points should not be used until their state is known.

Example

```
Private Declare Sub Sleep Lib "kernel32" (ByVal
dwMilliseconds As Long)

…

Dim point As GENCLIENTWRAPPERLib.DataPoint

Set point =
genclient.RequestDataPoint("ICONICS.Simulator\SimulatePLC.
Sine", 50, 0)


While (point.State = GC_POINT_UNKNOWN)

Sleep (100) 'wait until the point state is known

Wend
```

Property Timestamp([pvMilliseconds]) As Date

Description

Property of DataPoint object. Contains current timestamp (time of the last update).

Example

```
MsgBox "Last update at: " + Str(point.TimeStamp)
```

Property Value As Variant

Description

Property of DataPoint object. Contains current value of the point. This property is writeable. Writing to it starts an asynchronous write operation.

Example

```
MsgBox "Point value is: " + Str(point.Value)
```

Property WritesPending As Long

Description

Property of DataPoint object. Contains number of pending write operations. The write operation can be invoked by writing to the Value property.

Example

```
While (point.WritesPending > 0)

Sleep (100) 'wait for the writes to finish

Wend
```

Property LastError As Long

Property PointLifetime As Long

Property ReadTimeout As Long

Property ScanRate As Long

Property WriteTimeout As Long

Property Enabled As Long

Description

Property of SecurityPoint object. Indicates if security for the object represented by the SecurityPoint is enabled or disabled for the user currently logged in.

Example

```
Dim secpoint As GENCLIENTWRAPPERLib.SecurityPoint

Set secpoint =
genclient.RequestSecurityPoint(GC_SECURITY_FILENAME,
"Display1.gdf", 0)

…

If Not secpoint.Enabled Then MsgBox "Access to Display1 is
disabled"
```

Property State As GC_POINT_STATE

GENESIS32 Tool Tray OLE Automation Reference

OLE Automation support was originally integrated into GenTray in Version 6.0. The **IscriptCommands** interface allows Visual Basic and other clients to start and stop individual GENESIS32 applications as well as make changes to the existing GenTray configuration.

Methods

GetRegisteredApps(AppRefIDs, AppDescriptions)

Inputs

none

Outputs

AppRefIDs is a string array of reference IDs that are used to access individual components.

AppDescriptions is a string array of descriptions that correspond to the AppRefIDs.

Description

Obtains a list of registered GenTray applications. The AppRefIDs are the identifiers to be used to retrieve other information as described below.

Startup()

Inputs

none

Outputs

none

Description

Starts all components currently configured to AutoStart.

Shutdown()

Inputs

none

Outputs

none

Description

Shuts down all components currently configured to AutoStop.

Start(AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Outputs

none

Description

Starts the specified component.

Stop(AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Outputs

none

Description

Shuts down the specified component.

NOTE: A deadlock will occur if a script is used to run the GenTray **Stop(AppRefID as String)** method because the script will attempt to stop itself (e.g. if GraphWorX32 is used to run a script that instructs GenTray to stop GraphWorX32).

GetStartTableEntry(AppRefID, Position, RelativeStartDelay)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Outputs

Position: Long value that specifies the position in the startup table.

RelativeStartDelay: Long value specifying time in seconds to wait before starting application.

Description

Gets the current start table entries as configured in GenTray.

SetStartTableEntry(AppRefID, Position, RelativeStartDelay)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Position: Long value that specifies the position in the startup table.

RelativeStartDelay: Long value specifying time in seconds to wait before starting application.

Description

Sets the current start table entries in GenTray.

GetStopTableEntry (AppRefID, Position, RelativeStoptDelay)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Outputs

Position: Long value that specifies the position in the startup table.

RelativeStartDelay: Long value specifying time in seconds to wait before starting application.

Description

Gets the current stop table entries as configured in GenTray.

SetStopTableEntry (AppRefID, Position, RelativeStoptDelay)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Position: Long value that specifies the position in the startup table.

RelativeStartDelay: Long value specifying time in seconds to wait before starting application.

Description

Sets the current stop table entries in GenTray.

Properties

AutoStartState(AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Description

Boolean value reflects the specified application's AutoStart state.

AutoStopState(AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Description

Boolean value reflects the specified application's AutoStop state.

ServiceState(AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Description

Boolean value reflects the specified application's Service state.

AutomaticState(AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Description

Boolean value reflects the specified application's Automatic state when configured as a Service.

RunsAsService(AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Description

Boolean value reflects whether the specified application can run as a service.

IsRunning(AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Description

Boolean value reflects the specified application's current runtime status.

SupportsCommand(AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Description

Boolean value reflects whether the specified application supports a command line string.

StartCommand (AppRefID)

Inputs

AppRefID: String specifying application identifier obtained using GetRegisteredApps().

Description

String value reflects the specified application's command line string.

GraphWorX32 OLE Automation Reference

OLE Automation is used to access properties and methods of objects in GraphWorX32.

Properties are used to reference attributes if an object (for example, a GraphWorX32 display) has a *BackgroundColor* property. Methods are used to make the object perform an action (for example, a display has a *FileOpen* method that loads a new display).

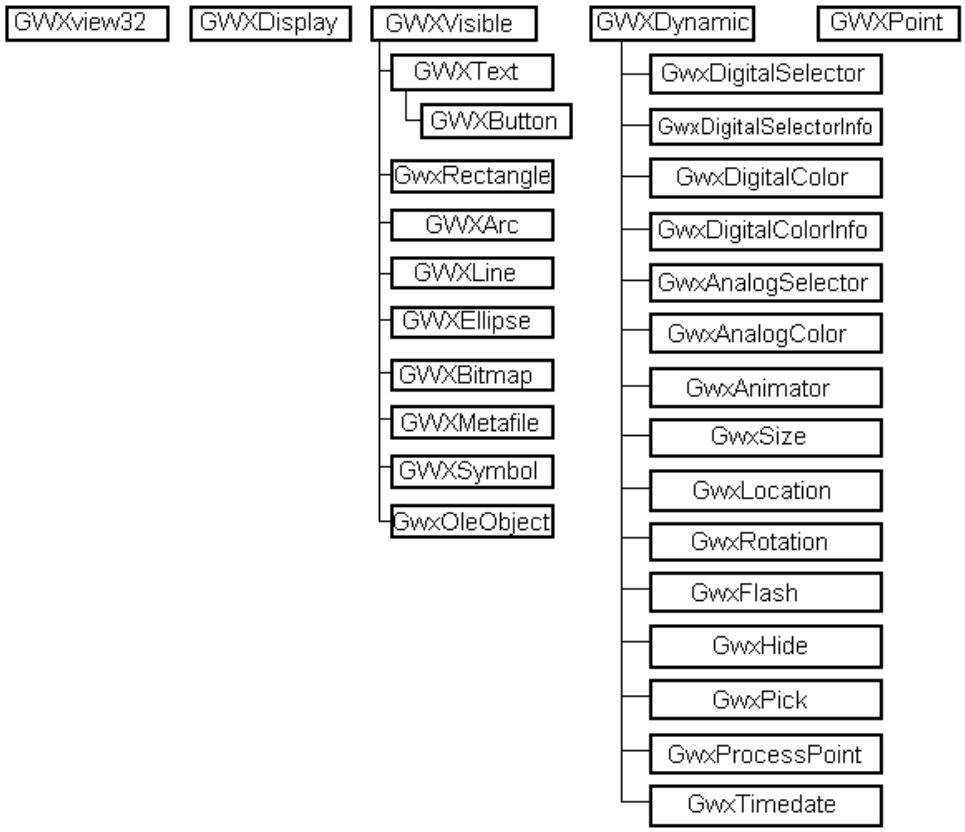
Properties and methods can be called from Visual Basic for Applications (VBA), VBScript, or JavaScript, or from C++ programs.

There are numerous object types in GraphWorX32, each with its own methods and properties. This section describes in detail, the methods and properties for the object types exposed by GraphWorX32.

Note: The GraphWorX32 automation interface is designed to be accessed from a single application. Accessing the GraphWorX32 automation interface from multiple external applications running simultaneously might lead to unpredictable results.

Object Hierarchy

The following diagram outlines the hierarchy of objects exposed by GraphWorX32 via OLE Automation.



GWXview32 is the data type of ICONICS' GWXview32 ActiveX control. For developers using ICONICS' ActiveX ToolWorX, **GWXview32** is the data type of

the base ActiveX control from which new controls are derived (new controls can inherit the properties and methods of GWXview32). Each GWXview32 object contains a single GwxDisplay object.

GwxDisplay is the object type for GraphWorX32 displays. A GraphWorX32 display contains collections of **GwxVisible** derived objects, **GwxDynamic** derived objects, and **GwxPoint** objects. The **GwxDisplay** object exposes methods for creating and getting existing instances of these visible and dynamic objects.

GwxVisible is the base object type for objects in a GraphWorX32 display that can be seen. All visible GraphWorX32 objects (i.e. **GwxRectangle**, **GwxEllipse**, **GwxText**, etc.) are derived from **GwxVisible** and consequently inherit all the properties and methods of **GwxVisible**.

GwxDynamic is the base object type for objects in a GraphWorX32 display that perform a dynamic transformation on an associated visible object. For example, a **GwxSize** object associated with a **GwxRectangle** object would change the size of the rectangle based on the OPC data value associated with the **GwxSize** object. All dynamic GraphWorX32 objects (i.e. **GwxSize**, **GwxLocation**, **GwxRotation**, etc.) are derived from **GwxDynamic** and consequently inherit all the properties and methods of **GwxDynamic**. Dynamic objects are not visible themselves; they exist to act upon the visual properties of an associated **GwxVisible** derived object. A **GwxVisible** object can have many **GwxDynamic** objects associated with it, by a **GwxDynamic** object has only one associated **GwxVisible** object.

GwxPoint is the base object type for objects in a GraphWorX32 display that reference OPC data, expressions, local variables, or constant values. Each **GwxDynamic** object may have one or more **GwxPoint** objects associated with it. **GwxPoint** objects are created and destroyed by **GwxDynamic** objects as needed.

Automation Programming in C++ and Visual Basic: Understanding Inheritance

The GraphWorX32 OLE Automation object hierarchy makes extensive use of inheritance. This raises some programming issues in both C++ and Visual Basic. To understand these programming issues, you must first understand how methods and properties are accessed via OLE Automation interfaces.

An OLE Automation method or property can be accessed by the method/property name or by an ID number associated with the method/property. When derived objects use inherited methods/properties from a base object, the ID numbers for the inherited methods/properties in the derived object must be changed such that the high word of the ID number is set to the level of inheritance of the derived object from base object in which the method/property is defined. For example, **GwxVisible** defines the **FillColor** property to have an ID number of **0x3. GwxRectangle** inherits the **FillColor** property as having the ID number **0x10003** (because **GwxRectangle** is derived from **GwxVisible**). Similarly, **GwxButton** inherits the **FillColor** property as having the ID number **0x20003** (because **GwxButton** is derived from **GwxText** which is derived from **GwxVisible**).

The nature of ID numbering for inheritance does not cause any problems when accessing objects via their actual object types. Programming issues arise when you want to downcast a derived object type to a base object type. For example, the **GwxDisplay** method **GetVisibleObjectFromName** returns a **GwxVisible** derived object, but there may be no way to know what the actual object type is ahead of time, so the programmer would want to assign the return value to a **GwxVisible** object (rather than a specific object type like **GwxRectangle** or **GwxEllipse**). But, if you then try to access methods/properties defined for **GwxVisible** of the returned object, they will not work because the ID numbers will be wrong (if the returned object was actually a **GwxRectangle** the ID for **FillColor** would be **0x10003**, but since the object is being accessed via the base **GwxVisible** object type it will try to use the ID number **0x3**).

In Visual Basic, the solution to this issue is simple. If you do not know the actual type of an object and you want to access it generically, use the object type **Object.** This will cause VB to access methods and properties of the object by name rather than by ID, thereby avoiding ID numbering issues.

In C++ this issue can require more work to get around. Typically when using OLE Automation objects in C++ you will allow Developer Studio to generate some kind of wrapper code for the automation interface. For instance, you can use Class Wizard to generate a **COleDispatchDriver** derived wrapper class. Or, you can use the **#import** directive to generate interface wrapper code (**TLI** and **TLH** files). Both these techniques however, access methods and properties via hard-coded ID numbers. Hence, when using such wrapper classes you will not be able to correctly downcast object types. The best way around this issue is to modify the generated wrapper classes for **GwxVisible** and **GwxDynamic** to access methods and properties by name rather than by hard-coded ID.

GraphWorX32 OLE Automation Example Displays

The GENESIS32 installation contains several example GraphWorX32 (.gdf) displays that demonstrate the use of OLE automation functions. Each of the functions for VBA or for VBScript is described in several displays as a part of a sample code. These displays are divided into two categories. The first one treats all aspects of the VBA programming for the GraphWorX. The second one treats the VBScript programming for GraphWorX. The displays describe the object programming in use, along with the help documentation.

There are eight main groups of displays, which are used as a source of code samples for the complete help for GraphWorX32.

- GwxdisplaycreateVBA.gdf - Contains functions and properties related to create functions. It also describes functions and properties of the class GwxVisible. The alphabetical list of these functions is in the Gwxdisplaycreate.txt file.
- GwxDisplayFileVBA.gdf, GwxDisplayFileVB.gdf - Contain functions and properties related to operations with files. The alphabetical list of these functions is in the GwxDisplayFileVB(A).txt files.
- GwxDisplayMiscVBA.gdf, GwxDisplayMiscVB.gdf - Contain miscellaneous functions and properties that should be described in one file together because of their different purpose and their number. The alphabetical list of these functions is in the GwxDisplayMiscVB(A).txt files.
- GwxDisplayObjectsVBA.gdf, GwxDisplayObjectsVB.gdf - Contain functions and properties related operations with objects. The complete list of these functions is in the GwxDisplayObjectsVB(A).txt files.
- GwxDisplayWindowVBA.gdf, GwxDisplayWindowVB.gdf - Contain functions and properties related to using functions operating with windows and runtime / design mode. The complete list of these functions is in the GwxDisplayWindowVB(A).txt files.
- GwxDisplayWindow2VBA.gdf, GwxDisplayWindow2VB.gdf - Contain functions and properties related to using functions operating with windows and runtime / design mode. The complete list of these functions is in the GwxDisplayWindow2VB(A).txt files.
- GwxDynamicVBA.gdf, GwxDynamicVB.gdf - Contain functions and properties related to operations with the GwxDynamic objects. The alphabetical list of these functions is in the GwxDynamicVB(A).txt files.
- GwxDisplayAlaisesVBA.gdf, GwxDisplayAlaisesVB.gdf - Describe usage of aliases. The list of these functions is in the GwxDisplayAlaisesVB(A).txt files.

The example displays contain the following OLE automation functions, which are described in detail in the sections below.

GwxDisplayAlaisesVB.gdf

- SetAliasDefinition (GwxDisplay)
- SetAliases (GwxDisplay)
- SetAlaisesFromFile (GwxDisplay)
- OpenSetAliasesFromFile (GwxDisplay)

- OpenSetAliasesDialog (GwxDisplay)
- GetLocalAliases (GwxDisplay)
- FileOpenSetAliases (GwxDisplay)
- OpenPopupWinSetAliases (GwxDisplay)
- OpenEmbeddedWinSetAliases (GwxDisplay)
- GetGASThemes (GwxDisplay)
- SetGasThemes (GwxDisplay)
- SetAliasDefinition (GwxVisible)
- SetAlaisesFromFile (GwxVisible)
- SetAliases (GwxVisible)
- SetAliases (GwxDynamic)
- SetAlaisesFromFile (GwxDynamic)
- SetAliasDefinition (GwxDynamic)
- GetAliasDefinition (GwxDynamic)

GwxDisplayAliasesVBA.gdf

- SetAliasDefinition (GwxDisplay)
- SetAliases (GwxDisplay)
- SetAlaisesFromFile (GwxDisplay)
- OpenSetAliasesFromFile (GwxDisplay)
- OpenSetAliasesDialog (GwxDisplay)
- GetLocalAliases (GwxDisplay)
- FileOpenSetAliases (GwxDisplay)
- OpenPopupWinSetAliases (GwxDisplay)
- OpenEmbeddedWinSetAliases (GwxDisplay)
- GetGASThemes (GwxDisplay)
- SetGasThemes (GwxDisplay)
- SetAliasDefinition (GwxVisible)
- SetAlaisesFromFile (GwxVisible)
- SetAliases (GwxVisible)
- GetLanguageAliasString (GwxDynamic)
- SetLanguageAliasString (GwxDynamic)
- SetAliases (GwxDynamic)
- SetAlaisesFromFile (GwxDynamic)
- SetAliasDefinition (GwxDynamic)
- GetAliasDefinition (GwxDynamic)

GwxDisplayCreateVB.gdf

- CreateAnalogColorDynamic
- CreateAnalogSelector
- CreateAnimator
- CreateArc

- CreateBitmap
- CreateButton
- CreateRectangle
- CreateDigitalSelector
- CreateEllipse
- CreateFlashDynamic
- CreateHideDynamic
- CreateLayer
- CreateRectangle
- CreateMetafile
- CreatePickDynamic
- CreatePolyline
- CreateProcessPoint
- CreateRectangle
- CreateRotationDynamic
- CreateSizeDynamic
- CreateStateField
- CreateSymbol
- CreateText
- CreateTiemdate

GwxDisplayFileVB.gdf

- CloseWindow
- DisplayBack
- DisplayForward
- ExportBitmapCaptureOfView
- FileNew
- FileOpen
- FileOpenSetAliases
- FilePrint
- FileSave
- FileSaveAs
- FileSaveAsType
- GetFileContaintsVBA
- GetFileName
- GetFilePath
- GetFileVersion
- GetPrintViewArea
- GetPrintWhiteBackground
- GetWorkingDirectory
- OpenEmbeddedWinSetAliases
- SetAliasesFromFile

- SetFileVersion
- SetPrintwhiteBackground
- SetWorkingDirectory

GwDisplayFileVBA.gdf

- FileNew
- FileOpen
- FilePrint
- FileSave
- GetFileContaintsVBA
- GetFileName
- FileSaveAsType
- FileOpenSetAliases
- FileSaveAs
- GetFilePath
- GetFileVersion
- SetFileVersion
- SetWorkingDirectory
- GetWorkingDirectory
- DisplayBack
- DisplayForward
- SetAliasesFromFile
- SetPrintwhiteBackground
- GetPrintWhiteBackground
- GetPrintViewArea
- ExportBitmapCaptureOfView
- OpenEmbeddedWinSetAliases
- CloseWindow

GwxDisplayMiscVB.gdf

- GetLocalAliases
- GetOpenPopupByIndex
- GetOpenPopupCount
- GetVCRMode
- SetVCRMode
- VCRControlPanel
- LogicalPointToClient
- LogicalPointToScreen
- OpenTagBrowser
- OpenPopupWinSetAliases
- GetSimulationvariables
- GetLocalVariables

- SetAliasDefinition
- SetAliases
- RepalceHostEx
- ReplaceHost
- GetLoadTab
- OpenSetAliasesDialog
- ReplaceStringInString
- SetSystemWideLanguage

GwxDisplayMiscVBA.gdf

- RemoveLayer
- CreateLayer
- DuplicateSelection
- GetEventScriptText
- GetLoadTab
- GetLocalAliases
- GetLocalVariables
- GetOpenPopupByIndex
- GetOpenPopupCount
- GetSimulationVariables
- GetSystemWideLanguage
- GwxGetVBAProject
- GetVCRMode
- LogicalPointToClient
- LogicalPointToScreen
- MoveSelectionBackward
- MoveSelectionForward
- MoveSelectionToBack
- MoveSelectionToFront
- OpenPopupWinSetAliases
- OpenSetAliasesDialog
- OpenTagBrowser
- PopAllCurrentSymbol
- PushCurrentSymbol
- QueryDataType
- QueryRanges
- RemoveLayer
- ReplaceHostEx
- ReplaceHost
- SetAliasDefinition
- SetAliases
- SetLayerOverrideHide

- SetModifiedFlag
- SetSystemWideLanguage
- SetVCRMode
- TestCustomSecurityItem
- UngroupSymbol
- VCRControlPanel

GwxDisplayObjectsVB.gdf

- GetObjectTypeName
- GetPointObjectFromName
- GetVisibleObjectFromIndex
- GetVisibleObjectFromName
- GetVisibleObjectFromPosition

GwxDisplayObjectsVBA.gdf

- ConvertGwxSymbolToLibraryObject
- DeleteObject
- DeselectAllObjects
- GetHeadObject
- GetObjectTypeName
- GetPointObjectFromName
- GetVisibleObjectFromIndex
- GetVisibleObjectFromName
- GetVisibleObjectFromPosition
- InsertLibraryObject
- InsertLibraryObjectByIndex
- InsertLibraryObjectByName
- InsertOleObject
- SetHeadObject

GwxDisplayWindow2VB.gdf

- ViewHome
- ViewUnzoom
- ViewPan
- ViewZoomPercent
- ViewFitToWindow
- IsModified
- ViewShowWholeDisplay
- ViewBoxZoom
- ViewZoomCustomDialog
- GetFrameWindowHandle
- EnableRuntimeZoomAndPan

- IsEnabledRuntimeZoomAndPan
- DisplayBack
- DisplayForward
- OpenEmbeddedWindow
- IsEmbeddedPopupWindow
- GetClientDimensionsPixels
- GetNumberOfTopLevelVisibles
- OpenPopupWindow
- GetParentDislay
- BackgroundColor
- CloseWindow

GwxDisplayWindow2VBA.gdf

- BackgroundColor
- CloseWindow
- DisplayBack
- DisplayForward
- EnableRuntimeZoomAndPan
- GetClientDimensionsPixels
- GetFrameWindowHandle
- GetNumberOfTopLevelVisibles
- GetParentDislay
- IsEmbeddedPopupWindow
- IsEnabledRuntimeZoomAndPan
- IsModified
- OpenEmbeddedWindow
- OpenPopupWindow
- ViewBoxZoom
- ViewFitToWindow
- ViewHome
- ViewPan
- ViewShowWholeDisplay
- ViewUnzoom
- ViewZoomCustomDialog
- ViewZoomPercent

GwxDisplayWindowVB.gdf

- ShowWindow
- RefreshWindow
- BringWindowToTop
- HideWindow
- StartRuntime

- StopRuntime
- ToggleRuntime
- ExitApplication
- MinimizeWindow
- MaximizeWindow
- RestoreWindow
- GetDisplayDimensions2
- SetDisplayDimensions
- SetViewDimensions
- GetViewDimensions2
- IsRuntimeMOde
- SetWindowDimensionsPixels
- SetWindowDimensionsPercent
- GetWindowDimensionsPercent2
- FetWindowDimensionsPixels2
- isPopupWindow
- OpenPopupWindow
- CloseAllPopupWindow
- closePopupwithTitle
- CloseWindow

GwxDynamicVB.gdf

- dataSource
- highRange
- lowRange
- ObjectName
- RangeOverride
- RequestDataType
- TimeRate
- UserCustomData
- UserDescriptuon
- GetAliasdefinition
- GetdataSourcePointObject
- GetHighRangePointObject
- GetLowRangePointObject
- GetNumberOfSubDynamic
- GetObjectTypeName
- GetSubDynamicObjectFromIndex
- GetVisibleObject
- SetAliasDefinition
- SetAliases
- SetAliasesFromFile

GwxDisplayWindowVBA.gdf

- ShowWindow
- RefreshWindow
- BringWindowToTop
- HideWindow
- StartRuntime
- StopRuntime
- ToggleRuntime
- ExitApplication
- MinimizeWindow
- MaximizeWindow
- RestoreWindow
- GetDisplayDimensions2
- SetDisplayDimensions
- SetViewDimensions
- GetViewDimensions2
- IsRuntimeMOde
- SetWindowDimensionsPixels
- SetWindowDimensionsPercent
- GetWindowDimensionsPercent2
- FetWindowDimensionsPixels2
- isPopupWindow
- OpenPopupWindow
- CloseAllPopupWindow
- closePopupwithTitle
- CloseWindow

GwxDynamicVBA.gdf

- dataSource
- GetAliasdefinition
- GetdataSourcePointObject
- GetHighRangePointObject
- GetLowRangePointObject
- GetNumberOfSubDynamic
- GetObjectTypeName
- GetSubDynamicObjectFromIndex
- GetVisibleObject
- highRange
- lowRange
- ObjectName
- RangeOverride
- RequestDataType

- SetAliasDefinition
- SetAliases
- SetAliasesFromFile
- TimeRate
- UserCustomData
- UserDescription

Events

This section describes the events that are exposed to GraphWorX32's integrated Visual Basic for Applications scripting. These events are also exposed from the GWXview32 ActiveX control.

Note: for developers using ActiveX ToolWorX, there are virtual member functions which correspond to the events described below.

Event DataEntryActivated

```
Event DataEntryActivated(DataEntry As Object)
```

This event is fired when the user activates a data entry object (i.e. clicks on a data entry object to type in a new value). The parameter **DataEntry** is the **GwxProcessPoint** object that caused the event.

Event DataEntryDeactivated

```
Event DataEntryDeactivated(DataEntry As Object)
```

This event is fired when the user deactivates a data entry object (i.e. the user clicks away from a data entry or uses the escape key to cancel editing, or after the user enters a value). The parameter **DataEntry** is the **GwxProcessPoint** object that caused the event.

Event DataEntryValueEntered

```
Event DataEntryValueEntered(DataEntry As Object)
```

This event is fired when the user enters a new data value into a data entry object. The parameter **DataEntry** is the **GwxProcessPoint** object that caused the event.

Event DialBeginDrag

```
Event DialBeginDrag(Dial As Object)
```

This event is fired when the user clicks the mouse down on a Gwx32 dial. The parameter **Dial** is the **GwxRotation** object of the dial that caused the event.

Event DialDrag

```
Event DialDrag(Dial As Object)
```

This event is fired when the user drags a Gwx32 dial (i.e. moving the mouse while the mouse button is held down on the dial). The parameter **Dial** is the **GwxRotation** object of the dial that caused the event.

Event DialEndDrag

```
Event DialEndDrag(Dial As Object)
```

This event is fired when the user releases the mouse button on a Gwx32 dial. This event is also called when the user turns the dial with the arrow keys instead of the mouse. The parameter **Dial** is the **GwxRotation** object of the dial that caused the event.

Event DisplayKeyDown

```
Event DisplayKeyDown(KeyCode As Long, Shift As Long, CancelProcessing)
```

Event DisplayKeyUp

```
Event DisplayKeyUp(KeyCode As Long, Shift As Long, CancelProcessing)
```

The above events are fired when a key is pressed (DisplayKeyDown) or released (DisplayKeyUp). It is possible to suppress the processing of the events by GraphWorX32 by setting CancelProcessing to True.

Note: This event will be fired only in the case it is enabled either programmatically (EnableRuntimeEvents) or manually in the **Format > Application Preferences > VBA** tab.

Parameters:

KeyCode

Key code, such as **vbKeyHome** (the HOME key).

Shift

Integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event.

CancelProcessing

Variant that can suppress further processing of the event by GraphWorX32 if set to True.

The following table shows the values for the *Shift* constants.

Constant	Value	Description
VbShiftMask	**1**	SHIFT key bit mask.
VbCtrlMask	**2**	CTRL key bit mask.
VbAltMask	**4**	ALT key bit mask.

Event DisplayLoad

```
Event DisplayLoad()
```

This event is fired just after a display is loaded into GraphWorX32.

Event DisplayMouseDblClick

```
Event DisplayMouseDblClick(Button As Long, Shift As Long, _
     X As Single, Y As Single, CancelProcessing)
```

Event DisplayMouseDown

```
Event DisplayMouseDown(Button As Long, Shift As Long, X As Single, _
     Y As Single, CancelProcessing)
```

Event DisplayMouseMove

```
Event DisplayMouseMove(Button As Long, Shift As Long, X As Single, _
     Y As Single, CancelProcessing)
```

Event DisplayMouseUp

```
Event DisplayMouseUp(Button As Long, Shift As Long, X As Single, _
     Y As Single, CancelProcessing)
```

The above events are fired when a mouse button is double-clicked (DisplayMouseDblClick), pressed (DisplayMouseDown), or released (DisplayMouseUp), or the mouse position is changed (DisplayMouseMove). It is possible to suppress the processing of the event by GraphWorX32 by setting CancelProcessing to True.

Note: This event will be fired only in the case it is enabled either programmatically (EnableRuntimeEvents) or manually in the **Format > Application Preferences > VBA** tab.

Parameters:

Button

An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. It indicates the complete state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are pressed.

Shift

An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT

key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of *shift* would be 6.

x, y

A number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

CancelProcessing

Variant that can suppress further processing of the event by GraphWorX32 if set to True.

Event DisplayUnload

```
Event DisplayUnload()
```

This event is fired just before a display is unloaded from GraphWorX32 (i.e. just prior to loading a new display into GraphWorX32 or prior to closing GraphWorX32).

Event LanguageChanged

```
Event LanguageChanged(long LocaleID)
```

This event is fired when GraphWorX32 is switched to a new language. The parameter **localeID** is the locale ID of the new language. *(Not available for displays running in GWXview32 objects*

Event PickPostDown

```
Event PickPostDown(Pick As Object)
```

This event is fired when the user presses down on a Gwx32 pick action. The event is fired after the pick's "OnDown" action is executed. The parameter Pick is **GwxPick** object of the Pick dynamic that caused the event.

Event PickPostUp

```
Event PickPostUp(Pick As Object)
```

This event is fired when the user releases (mouse button up) a Gwx32 pick action. The event is fired after the pick's "OnUp" action is executed. The parameter Pick is **GwxPick** object of the Pick dynamic that caused the event.

Event PickPostWhileDown

```
Event PickPostWhileDown(Pick As Object)
```

This event is fired while the pick action is held down at time interval specified in the pick object. The event is fired after the pick's "WhileDown" action is executed. The parameter Pick is **GwxPick** object of the Pick dynamic that caused the event.

Event PickPreDown

```
Event PickPreDown(Pick As Object)
```

This event is fired when the user presses down on a Gwx32 pick action. The event is fired before the pick's "OnDown" action is executed. The parameter Pick is **GwxPick** object of the Pick dynamic that caused the event.

Event PickPreUp

```
Event PickPreUp(Pick As Object)
```

This event is fired when the user releases (mouse button up) a Gwx32 pick action. The event is fired before the pick's "OnUp" action is executed. The parameter Pick is **GwxPick** object of the Pick dynamic that caused the event.

Event PickPreWhileDown

```
Event PickPreWhileDown(Pick As Object)
```

This event is fired while the pick action is held down at time interval specified in the pick object. The event is fired before the pick's "WhileDown" action is executed. The parameter Pick is **GwxPick** object of the Pick dynamic that caused the event.

Event PostAnimateDisplay

```
Event PostAnimateDisplay()
```

This event is fired after GraphWorX32 has finished animating a display.

Event PostAnimateLayer

```
Event PostAnimateLayer(LayerName As String)
```

This event is fired after GraphWorX32 has finished animating the specified layer.

Event PostDeanimateDisplay

```
Event PostDeanimateDisplay()
```

This event is fired after GraphWorX32 has finished deanimating a display.

Event PostDeanimateLayer

```
Event PostDeanimateLayer(LayerName As String)
```

This event is fired after GraphWorX32 has finished deanimating the specified layer.

Event PostRuntimeStart

```
 Event PostRuntimeStart()
```

This event is fired when GraphWorX32 has finished entering runtime mode.

Event PostRuntimeStop

```
Event PostRuntimeStop()
```

This event is fired after GraphWorX32 has completely exited runtime mode.
void PreAnimateDisplay()

This event is fired just before GraphWorX32 animates a display (animating a display is the process of requesting the tags from the OPC server(s)). Displays are animated when loaded during runtime mode, and deanimated when unloaded during runtime mode.

Event PreAnimateDisplay

```
Event PreAnimateDisplay()
```

This event is fired just before GraphWorX32 animates the specified display (animating a display is the process of requesting the tags from the OPC server(s) that are referenced in the display).

Event PreAnimateLayer

```
Event PreAnimateLayer(LayerName As String)
```

This event is fired just before GraphWorX32 animates the specified display layer (animating a layer is the process of requesting the tags from the OPC server(s) that are referenced in the layer).

Event PreDeanimateDisplay

```
Event PreDeanimateDisplay()
```

This event is fired just before GraphWorX32 deanimates a display (deanimating a display is the process of releasing the tags previously requested from the OPC server(s)).

Event PreDeanimateLayer

```
Event PreDeanimateLayer(LayerName As String)
```

This event is fired just before GraphWorX32 deanimates the specified display layer (deanimating a layer is the process of releasing the tags previously requested from the OPC server(s)).

Event PreRuntimeStart

```
Event PreRuntimeStart()
```

This event is fired just before GraphWorX32 is put into runtime mode.

Event PreRuntimeStop

```
Event PreRuntimeStop()
```

This event is fired just before GraphWorX32 exits runtime mode.

Event SliderBeginDrag

```
Event SliderBeginDrag(Slider As Object)
```

This event is fired when the user clicks the mouse down on a Gwx32 slider. The parameter **Slider** is the **GwxLocation** object of the slider that caused the event.

Event SliderDrag

```
Event SliderDrag(Slider As Object)
```

This event is fired when the user drags a Gwx32 slider (i.e. moving the mouse while the mouse button is held down on the slider). The parameter **Slider** is the **GwxLocation** object of the slider that caused the event.

Event SliderEndDrag

```
Event SliderEndDrag(Slider As Object)
```

This event is fired when the user releases the mouse button on a Gwx32 slider. This event is also called when the user moves the slider with the arrow keys instead of the mouse. The parameter **Slider** is the **GwxLocation** object of the slider that caused the event.

Enumerations

Enumeration constants do not work in the VBScript editor. The corresponding numeric value must be used. For example the following line of code opens a display and sets the alises values by reading them from a file.

ThisDisplay.FileOpenSetAliases("myFileName.gdf", AliasSetFromFile, "AliasCommandString.txt")

In VBScript and JScript you must replace the enumeration constant AliasSetFromFile with its corresponding numerical value, which is 1. This is how the line will look in VBScript:

ThisDisplay.FileOpenSetAliases("myFileName.gdf", **1**, "AliasCommandString.txt")

GWX3DEDGESTYLE

```
helpstring("3D Edge Style")]
enum {
        EdgeNone = 0,
        EdgeRaised = 5,
        EdgeEtched = 6,
        EdgeBump   = 9,
        EdgeSunken = 10
} GWX3DEDGESTYLE;
```

GWXLINESTYLE

```
helpstring("Line Style")]
enum {
        LineSolid           = 0,
        LineDash            = 1,
        LineDot             = 2,
        LineDashDot         = 3,
        LineDashDotDot      = 4,
        LineNone            = 5
} GWXLINESTYLE;
```

GWXTEXTALIGNMENT

```
helpstring("Text Alignment")]
enum {
        TextAlignLeft    = 0,
        TextAlignCenter = 1,
        TextAlignRight   = 2
} GWXTEXTALIGNMENT;
```

GWXSIZEDYNTYPE

```
helpstring("Size Dynamic Direction")]
enum {
        SizeLeft            = 0,
        SizeRight           = 1,
        SizeUp              = 2,
```

```
        SizeDown            = 3,
        SizeUpLeft          = 4,
        SizeUpRight         = 5,
        SizeDownLeft        = 6,
        SizeDownRight       = 7,
        SizeLeftRight       = 8,
        SizeUpDown          = 9,
        SizeLeftRightBias   = 10,
        SizeUpDownBias      = 11,
        SizeAllFour         = 12,
        SizeLeftRightUp     = 13,
        SizeLeftRightDown   = 14,
        SizeUpDownLeft      = 15,
        SizeUpDownRight     = 16
} GWXSIZEDYNTYPE;
```

GWXPICKACTION

```
helpstring("Pick Action")]
enum {
        PickLoadDisplay     = 0,
        PickDragDropLoad    = 1,
        PickPopupWindow     = 2,
        PickDownloadValue   = 3,
        PickToggleValue     = 4,
        PickLaunchApp       = 5,
        PickClose           = 6,
        PickRunScript       = 7,
        PickCustomCommand   = 8,
        PickEmbeddedWindow  = 9,
        PickDisplayBack     = 10,
        PickDisplayForward  = 11,
        PickSetAliases      = 12,
        PickAliasesDialog   = 13,
        PickLayerHideOn     = 14,
        PickLayerHideOff    = 15,
        PickLayerHideToggle = 16,
        PickPopupUserMenu   = 17,
        PickSwitchLanguage  = 18,
        PickSelectThema     = 19,
        PickLaunchWebPage   = 20
} GWXPICKACTION;
```

GWXSCRIPTTYPE

```
helpstring("Script Type")]
enum {
        ScriptTypeNone      = -1,
        ScriptTypeVBA       = 0,
        ScriptTypeVBScript  = 1,
        ScriptTypeJScript   = 2
} GWXSCRIPTTYPE;
```

GWXEVENTNAME

```
helpstring("Event Name")]
enum {
        DataEntryActivated      = 0,
        DataEntryDeactivated    = 1,
        DataEntryValueEntered = 2,
        DialBeginDrag           = 3,
        DialDrag                = 4,
        DialEndDrag             = 5,
        DisplayLoad             = 6,
        DisplayUnload           = 7,
        LanguageChanged         = 8,
        PickPostDown            = 9,
        PickPostUp              = 10,
        PickPostWhileDown       = 11,
        PickPreDown             = 12,
        PickPreUp               = 13,
        PickPreWhileDown        = 14,
        PostAnimateDisplay      = 15,
        PostDeanimateDisplay    = 16,
        PreAnimateDisplay       = 17,
        PreDeanimateDisplay     = 18,
        PreAnimateLayer         = 19,
        PostAnimateLayer        = 20,
        PreDeanimateLayer       = 21,
        PostDeanimateLayer      = 22,
        PreRuntimeStart         = 23,
        PostRuntimeStart        = 24,
        PreRuntimeStop          = 25,
        PostRuntimeStop         = 26,
        SliderBeginDrag         = 27,
        SliderEndDrag           = 28,
        SliderDrag              = 29,
        VCRStart                = 30,
        VCRStop                 = 31,
        GlobalAliasChanged      = 32,
        ViewChanged             = 33,
        DisplayChanged          = 34,
        WindowChanged           = 35,
        PrePrint                = 36,
        PostPrint               = 37
} GWXEVENTNAME;
```

GWXALIASCOMMAND

```
helpstring("Alias Command")]
enum {
        AliasNoCommand      = 0,
        AliasSetFromFile    = 1,
        AliasSetDirect      = 2
} GWXALIASCOMMAND;
```

GWXBUTTONTYPE

```
helpstring("Button Type")]
```

```
enum {
        ButtonNormal            = 0,
        ButtonCheck             = 1,
        ButtonRadio             = 2
} GWXBUTTONTYPE;
```

GWXMOUSEBUTTON

```
helpstring("Mouse Button")]
enum {
        MouseButtonLeft         = 0,
        MouseButtonMiddle       = 1,
        MouseButtonRight        = 2
} GWXMOUSEBUTTON;
```

GWXEXECUTIONTRIGGER

```
helpstring("Execution Trigger")]
enum {
        TriggerOnDown                   = 1,
        TriggerWhileDown                = 2,
        TriggerOnDnWhileDn              = 3,
        TriggerOnUp                     = 4,
        TriggerOnDnOnUp                 = 5,
        TriggerWhileDnOnUp              = 6,
        TriggerOnDnWhileDnOnUp          = 7
} GWXEXECUTIONTRIGGER;
```

GWXTIMEDATEFORMATTYPE

```
helpstring("Timedate Format Type")]
enum {
        TimeOnly                = 0,
        DateOnly                = 1,
        TimeAndDate             = 2
} GWXTIMEDATEFORMATTYPE;
```

GWXDATATYPE

```
helpstring("Data Type")]
enum {
        DataTypeEmpty= 0,
        DataTypeShort  = 2,
        DataTypeLong   = 3,
        DataTypeFloat  = 4,
        DataTypeDouble= 5,
        DataTypeString = 8,
        DataTypeBool   = 11,
        DataTypeChar   = 16,
        DataTypeByte   = 17,
        DataTypeWord  = 18,
        DataTypeDword= 19
} GWXDATATYPE;
```

GWXSCALEMODE

```
helpstring("Scale Mode")]
enum {
        FixedScale                  = 0,
        Scaleable                   = 1,
        ScaleablePreserveAspect     = 2
} GWXSCALEMODE;
```

GWXARCTYPE

```
helpstring("Arc Type")]
enum {
        ArcArc          = 0,
        ArcPie          = 1,
        ArcChord        = 2
} GWXARCTYPE;
```

GWXGRADIENTSTYLE

```
helpstring("Gradient Style")]
enum {
        GradientHorizontal      = 0,
        GradientVertical        = 1,
        GradientSquare          = 2
} GWXGRADIENTSTYLE;
```

GWXGETFILEVERSION

```
helpstring("Get File Version")]
enum {
        OpenFileError           = -1,
        Ver520                  = 0,
        Ver600                  = 1,
        Ver600SP                = 2,
        VerMGraphics40          = 3,
        Ver610                  = 4,
        Ver620                  = 5,
        VerLatest               = 9999
} GWXGETFILEVERSION;
```

GWXSETFILEVERSION

```
helpstring("Set File Version")]
enum {
        SetVer600SP             = 2,
        SetVerMGraphics40       = 3,
        SetVer610               = 4,
```

```
        SetVerLatest            = 9999
} GWXSETFILEVERSION;
```

GWXSECURITYWRITESTATUS

```
helpstring("Get Security Write Status")]
enum {
        WriteUninitialized      = -1,
        WriteEnabled            = 0,
        WriteDisabled           = 1,
        WriteCritical           = 2
} GWXSECURITYWRITESTATUS;
```

Properties and Methods

GraphWorX32 includes the following general OLE Automation classes:

GwxView32

GwxDisplay

GwxVisible

GwxDynamic

GwxPoint

GWXView32

GWXview32 is the object type for the GraphWorX32 ActiveX control (GWXview32.ocx). When using ActiveX ToolWorX, the properties and methods described in this section can be inherited by new ActiveX controls. **GWXview32** objects can be created via the ProgID **GWXVIEW32.GWXview32Ctrl.1**.

Properties

AutoStartRuntime

```
AutoStartRuntime As Boolean
```

When TRUE, the object/control will automatically enter runtime mode when the object's container enters runtime mode. This property only works if the container supports the AmbientUserMode property.

HorizontalScrollbar

```
HorizontalScrollbar As Boolean
```

When TRUE, the horizontal scrollbar of the control's window is visible, when FALSE the horizontal scrollbar is hidden. This property is ignored if **OverrideScrollbarSettings** is FALSE.

OverrideScrollbarSettings

```
OverrideScrollbarSettings As Boolean
```

When TRUE, this property indicates the object/control will override the scrollbar visibility settings of the display currently loaded in the control. When FALSE, the object/control will use the scrollbar settings defined in the currently loaded display.

UseAmbientBackColor

```
UseAmbientBackColor As Boolean
```

When TRUE, the background color of the object/control will automatically be set to match the background color of the container in which this object is embedded. This property only works if the container supports the AmbientBackColor property.

VerticalScrollbar

```
VerticalScrollbar As Boolean
```

When TRUE, the vertical scrollbar of the control's window is visible, when FALSE the vertical scrollbar is hidden. This property is ignored if **OverrideScrollbarSettings** is FALSE.

DisplayName

```
DisplayName As String
```

File name of the GraphWorX32 display loaded in this object/control. Setting this property to a new file name will cause the new display to be loaded.

Appearance

```
Appearance As Integer
```

Appearance of the window border. Valid values are:

Flat = 0
3-D = 1

BorderStyle

```
BorderStyle As Integer
```

Border style of the window. Valid values are:

No border = 0
Normal Border = 1

Methods

Function GetDisplay

```
Function GetDisplay() As Object
```

Returns the **GwxDisplay** object for the display currently loaded in this object/control. You can then use this object to access all of the properties and methods of all the object types described in this section.

Function ReplaceFilePath

```
Function ReplaceFilePath(OldSubstring As String, _
     NewSubstring As String) As Long
```

Works over path name parameters of pick dynamics. It returns 0 on success and HRESULT when something fails.

VBA example:

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces file path in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceFilePath("C:\Windows\Temp", "D:\Temp")
'if Status <> 0 then there was no replacement performed or an error occurs

If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

Function ReplaceHost

```
Function ReplaceHost(OldHostName As String, NewHostName As String) _
     As Long
```

Works over both data source tags and path name attributes and replaces node name **substring** within URL path only. Returns 0 on success and HRESULT when something fails.

VBA example 1:

```
    If 0 <> ThisDisplay.ReplaceHost( _
            "\\host1\Simulator\SimulatePLC.Sine", _
            "\\host2\Simulator\SimulatePLC.Sine") _
            Then
        MsgBox "Something fails."
    Else
        MsgBox "The node name has been successfully replaced."
    End If
```

VBA example 2:

```
'similar to ReplaceTag
'VBA example, works only in configure mode
```

```
'replaces host name in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceHost("Host1", "Host2")
'if Status <> 0 then there was no replacement performed or an error occurs

If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

Function ReplaceHostEx

```
Function ReplaceHostEx(OldHostNameSubstring As String, _
      NewHostNameSubstring As String, MatchCase As Boolean, _
      MatchWholeWord As Boolean) As Long
```

Works over both data source tags and path name attributes and replaces node name **substring** within URL path only, and supports case-sensitivity, wildcard strings and MatchWholeWord flag. It returns 0 on success and HRESULT when something fails.

VBA example 1:

```
    If 0 <> ThisDisplay.ReplaceHostEx( _
            "\\localhost\ICONICS.Simulator.1\SimulatePLC.Sine", _
            "\\T1021\ICONICS.Simulator.1\SimulatePLC.Sine", _
            False, _
            False) _
        Then
        MsgBox "Something fails."
    Else
        MsgBox "The node name has been successfully replaced."
    End If
```

VBA example 2:

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces host name in whole display, regarding "case" and "whole_
'words" options

Dim Status As Long
Status = ThisDisplay.ReplaceHostEx("Host1", "Host2", True, True)
'if Status <> 0 then there was no replacement performed or an error occurs

If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

Function ReplaceTag

```
Function ReplaceTag(OldSubstring As String, NewSubstring As String) _
     As Long
```

Works over data source tags in dynamic actions of all display objects. Function Replaces **OldSubstring** with **NewSubstring** in certain situations. It returns 0 on success and HRESULT when something fails.

VBA example:

```
'VBA example, works only in configure mode
'replaces tag name in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceTag("gfwsim.ramp.float", _ "gfwsim.sine.double")
'if Status <> 0 then there was no replacement performed
'or an error occurs

If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

long ReplaceStringInString

```
long ReplaceStringInString (BSTR StringToReplace, _
     BSTR OldSubstring, BSTR NewSubsring)
```

Function replaces **OldSubstring** with **NewSubstring** inside (within) StringToReplace. It returns 0 on success and HRESULT when something fails.

Sub SelectLanguage

```
Sub SelectLanguage(Language As String, _
     LanguageSubset As String, LocaleID As Long, _
     SynchronizeLCID As Boolean)
```

Sub SetGasParentCookie

```
Sub SetGasParentCookie(newCookie As Long)
```

Used internally for communication between ActiveX control and container. Do not modify.

Sub StartRuntime

```
Sub StartRuntime()
```

Puts this object/control into runtime mode.

Sub StopRuntime

```
Sub StopRuntime()
```

Takes this object/control out of runtime mode.

GwxDisplay

GwxDisplay is the object type for GraphWorX32 displays. Properties and methods of the **GwxDisplay** object type are described in detail below. Also described, are the events exposed to GraphWorX32's integrated Visual Basic for Applications (VBA). **GwxDisplay** objects can be created via the ProgID **Gwx32.Display.**

Properties

BackgroundColor

```
BackgroundColor As OLE_COLOR
```

Gets/Sets the current display's background color.

VBA example:

```
Dim red As Byte
Dim green As Byte
Dim blue As Byte
Dim rgbOLE As OLE_COLOR

rgbOLE = ThisDisplay.BackgroundColor

red = 255
green = 255
blue = 0

ThisDisplay.BackgroundColor = rgb(red, green, blue)

MsgBox "Undo BacgroundColor"

ThisDisplay.BackgroundColor = rgbOLE
```

VB example:

```
Dim red
Dim green
Dim blue
Dim rgbOLE

rgbOLE = ThisDisplay.BackgroundColor

red = 255
green = 255
blue = 0

ThisDisplay.BackgroundColor = RGB(red, green, blue)

MsgBox "Undo BacgroundColor"

ThisDisplay.BackgroundColor = rgbOLE
```

IsEnabledRuntimeZoomAndPan

```
IsEnabledRuntimeZoomAndPan As Boolean
```

VBA example:

```
Dim flag As Boolean

MsgBox "Is runtime zoom and pan enabled?" + _
    Chr(13) + CStr(ThisDisplay.IsEnabledRuntimeZoomAndPan)
```

VB example:

```
MsgBox "Is runtime zoom and pan enabled?" + _
    Chr(13) + CStr(ThisDisplay.IsEnabledRuntimeZoomAndPan)
```

Redraw

```
Redraw As Boolean
```

When TRUE, automation calls when change the visual appearance of a display will automatically refresh the display. When FALSE, the user must explicitly refresh the display (for instance, you may want to change the attributes of many objects, and refresh them all at once).

```
'To run this script you need to create an object with _
'name "Test2".

'Toggle Redraw so the display will
'not be able to automatically
'refresh. At the end of this script
'we will manualy call RefreshWindow.
ThisDisplay.Redraw = False

'Get rectangle from name:
Dim oRec As GwxRectangle
Set oRec = ThisDisplay.GetVisibleObjectFromName("Test2")
oRec.Selected = True

'Set Redraw to True:
ThisDisplay.Redraw = True

'and finally use refresh:
Call ThisDisplay.RefreshWindow
```

VB example:

```
'This routine changes the dimensions of display with following _
'display recovery.


'Declare coordinates as long values
Dim widthDisplay
Dim heightDisplay

Dim widthDisplayMem
Dim heightDisplayMem

'Remember current dimensions:

Call ThisDisplay.GetDisplayDimensions2(widthDisplayMem, _
   heightDisplayMem)
widthDisplayMem = 1036
heightDisplayMem = 767

'Set up new parameters:
```

```
    widthDisplay = 600
    heightDisplay = 400

    'Set the display parameters:
    Call ThisDisplay.SetDisplayDimensions(widthDisplay, _
        heightDisplay)
    'and refresh the window:
    ThisDisplay.RefreshWindow

    MsgBox "When you click on OK the dimensions will be restored."

    'Again, set display parameters:
    Call ThisDisplay.SetDisplayDimensions(widthDisplayMem, _
       heightDisplayMem)

   'and once more refresh the window:
    ThisDisplay.RefreshWindow
```

SaveAsVBA

```
SaveAsVBA As Boolean
```

Save the display by default as VBA display (*TRUE*) or as non VBA display (*FALSE*). See the section “General → Save Display” in application Preferences.

Note: Entire code from the VBA editor will be lost whenever you save the display as non VBA (*SaveAsVBA = False*).

ScaleMode

```
ScaleMode As GWXSCALEMODE
```

Sets the scaling mode of the current display. Valid values are:

FixedScale = 0
Scaleable = 1
ScaleablePreserveAspect = 2

Methods

Sub BringWindowToTop

```
Sub BringWindowToTop()
```

It brings the GraphWorX32 main window to the top of the window z-order.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
    'Declare a new GwxDisplay
    Dim newGwxDisplay As New Gwx32.GwxDisplay

    'Show the display
    Call newGwxDisplay.ShowWindow

    Call newGwxDisplay.BringWindowToTop

    MsgBox "This function works properly only" + _
       " within a dialog which follows."
```

VB example:

```
'Declare a new GwxDisplay
Dim newGwxDisplay

MsgBox "New window is launched..."

'Assign an object to newGwxDisplay
Set newGwxDisplay = CreateObject("Gwx32.Display")

'Show the display...
Call newGwxDisplay.ShowWindow

'and bring it to top.
Call newGwxDisplay.BringWindowToTop
```

Sub ClearGradientFill

```
Sub ClearGradientFill()
```

Clears a previously set background gradient fill (restores solid background fill).

Sub CloseAllPopupWindows

```
Sub CloseAllPopupWindows()
```

Closes all currently open popup windows.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
'Close the window from he parent display.
ThisDisplay.CloseAllPopupWindows
```

VB example:

```
'Close the window from he parent display.
ThisDisplay.CloseAllPopupWindows
```

Function ClosePopupWithTitle

```
Function ClosePopupWithTitle(titleSubstring As String) As Boolean
```

Closes the popup window that has the specified substring as part or all of the text in the popup window's title bar. Returns TRUE if the specified popup window was successfully closed, FALSE otherwise.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
'Ignore trailing string "GraphWorx32" from Iconics
ThisDisplay.ClosePopupWithTitle ("PopupWindow")
```

VB example:

```
'Ignore trailing string "GraphWorx32" from Iconics
ThisDisplay.ClosePopupWithTitle ("PopupWindow")
```

Sub CloseWindow

```
Sub CloseWindow()
```

Closes the window for this display (this method is like the **ExitApplication** method, however **CloseWindow** also works for displays running in popup windows and embedded popup windows).

Note: Not available in ActiveX ToolWorX.

VBA example:

```
'Create a new instance from the parent display
'where you can use this function.
ThisDisplay.CloseWindow
```

VB example:

```
'Create a new instance from the parent display
'where you can use this function.
ThisDisplay.CloseWindow
```

Function ConvertGwxSymbolToLibraryObject

```
Function ConvertGwxSymbolToLibraryObject(gwxSymbol As Object, _
     libraryObject As Object) As Boolean
```

Converts GwxSymbol object into Symbol Library object. As such can be inserted into one of Symbol Library category files (.sdf) via Symbol Library automation.

(See the Symbol Library Help documentation for more information.)
Note: This function works in GraphWorX32 design mode only.

VBA example:

```
Dim newSym As Object
Dim libObj As Object

If Not ThisDisplay.InsertLibraryObjectByName("Misc",_
   "Check") Then
  MsgBox "Error with InsertLibraryObjectByName!"
End If

Set newSym = ThisDisplay.GetHeadObject

MsgBox newSym.GetObjectTypeName

If Not ThisDisplay.ConvertGwxSymbolToLibraryObject(newSym, _
        libObj) Then
  MsgBox "Error converting symbol to library object!"
Else
  MsgBox "GwxSymbol successfully converted to library object!"
End If
```

Function CreateAnalogColorDynamic

```
Function CreateAnalogColorDynamic(visibleObjectName As String, _
     dynamicObjectName As String, _
     changeFill As Boolean, _
     changeLine As Boolean, _
     changeShadow As Boolean, _
     startFillColor As OLE_COLOR, _
     endFillColor As OLE_COLOR, _
     startLineColor As OLE_COLOR, _
     endLineColor As OLE_COLOR,  _
```

```
    startShadowColor As OLE_COLOR, _
    endShadowColor As OLE_COLOR, _
    defaultColorAbove As Boolean, _
    defaultColorBelow As Boolean) As Object
```

Creates a **GwxAnalogColor** object with the specified attributes and attaches it to the visible object with the specified object name. Returns the newly created **GwxAnalogColor** object on success, NULL if the operation failed. (See also **GwxAnalogColor** and **GwxDynamic**).

Note: Available only in configure mode.

VBA example:

```
Dim objRect As GwxRectangle

'button
o.Selected = False

Call ThisDisplay.DeleteObject("TheRectangle")
Set objRect = ThisDisplay.CreateRectangle( _
                            442, _
                            213, _
                            80, _
                            60, _
                            True, _
                            RGB(100, 200, 230), _
                            RGB(100, 200, 230), _
                            1, _
                            LineNone, _
                            False, _
                            RGB(192, 192, 192), _
                            EdgeEtched, _
                            False, _
                            "TheRectangle", _
                            True)
If objRect Is Nothing Then
    MsgBox "Operation failed."
    Exit Sub
Else
    MsgBox "A new object has been created."
End If

objRect.UserDescription = "The Rectangle"


Dim obj As GwxAnalogColor

ThisDisplay.DeleteDynamic ("ObjAnalogColor")

Set obj = ThisDisplay.CreateAnalogColorDynamic( _
                    "TheRectangle", _
                    "ObjAnalogColor", _
                    True, _
                    True, _
                    True, _
                    RGB(223, 223, 223), _
                    RGB(100, 100, 200), _
                    RGB(223, 223, 223), _
                    RGB(100, 100, 200), _
                    RGB(223, 223, 223), _
                    RGB(10, 10, 200), _
                    True, _
```

```
                        True)

    If obj Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    obj.dataSource = "gfwsim.sine.double"
    obj.UserDescription = "GwxAnalogColor"
```

Function CreateAnalogSelector

```
Function CreateAnalogSelector(symbolName As String, _
     dynamicObjectName As String, _
     hiddenWhenAbove As Boolean, _
     hiddenWhenBelow As Boolean) As Object
```

Creates a **GwxAnalogSelector** object with the specified attributes and attaches it to the **GwxSymbol** object with the specified object name. Returns the newly created **GwxAnalogSelector** object on success, NULL if the operation failed.

Note: Available only in configure mode. See also **GwxAnalogSelector**, **GwxSymbol**, **GwxVisible**, and **GwxDynamic**.

VBA example:

```
    'button
    o.Selected = False

    Dim obj As GwxAnalogSelector

    ThisDisplay.DeleteDynamic ("ObjectDyn")

    Dim objRect As GwxRectangle
    Set objRect = ThisDisplay.GetVisibleObjectFromName( _
                            "Rectangle_0")
    objRect.Selected = True

    Dim objEll As GwxEllipse
    Set objEll = ThisDisplay.GetVisibleObjectFromName( _
                            "Ellipse_0")
    objEll.Selected = True

    Dim objCir As GwxEllipse
    Set objCir = ThisDisplay.GetVisibleObjectFromName( _
                            "Circle_0")
    objCir.Selected = True

    'Deselect the button:
    o.Selected = False

    'Use selected visibles to create a group:
    ThisDisplay.CreateSymbol ("Symbol_0")

    Set obj = ThisDisplay.CreateAnalogSelector( _
                    "Symbol_0", _
                    "ObjectDyn", _
                    False, _
                    False)

    If obj Is Nothing Then
```

```
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    obj.dataSource = "gfwsim.sine.double"
    obj.UserDescription = "GwxAnalogSelector"
```

Function CreateAnimator

```
Function CreateAnimator(symbolName As String, _
     dynamicObjectName As String, _
     animateWhenTrue As Boolean, _
     visibleWhenOff As Boolean, _
     currentFrameWhenOff As Boolean) As Object
```

Creates a **GwxAnimator** object with the specified attributes and attaches it to the **GwxSymbol** object with the specified object name. Returns the newly created **GwxAnimator** object on success, NULL if the operation failed.

Note: (See also GwxAnimator, GwxSymbol, GwxVisible, and GwxDynamic).
Available only in configure mode.

VBA example:

```
    'button
    o.Selected = False

    ThisDisplay.DeleteDynamic ("ObjectDyn")

    Dim objRect As GwxRectangle
    Set objRect = ThisDisplay.GetVisibleObjectFromName( _
                            "Rectangle_0")
    objRect.Selected = True

    Dim objEll As GwxEllipse
    Set objEll = ThisDisplay.GetVisibleObjectFromName( _
                            "Ellipse_0")
    objEll.Selected = True

    Dim objCir As GwxEllipse
    Set objCir = ThisDisplay.GetVisibleObjectFromName( _
                            "Circle_0")
    objCir.Selected = True

    'Deselect the button:
    o.Selected = False

    'Use selected visibles to create a group:
    ThisDisplay.CreateSymbol ("Symbol_0")

    Dim obj As GwxAnimator
    Set obj = ThisDisplay.CreateAnimator( _
                        "Symbol_0", _
                        "ObjectDyn", _
                        True, _
                        True, _
                        False)

    If obj Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
```

```
    Else
        MsgBox "A new object has been created."
    End If

    obj.dataSource = "gfwsim.sine.double"
    obj.UserDescription = "GwxAnimator"
```

Function CreateArc

```
Function CreateArc(arcType As GWXARCTYPE, _
     centerX As Single, _
     centerY As Single, _
     radiusX As Single, _
     radiusY As Single, _
     startAngle As Single, _
     endAngle As Single, _
     isFilled As Boolean, _
     fillColor As OLE_COLOR, _
     lineColor As OLE_COLOR, _
     lineWidth As Long, _
     lineStyle As GWXLINESTYLE, _
     hasShadow As Boolean, _
     shadowColor As OLE_COLOR, _
     edgeStyle As GWX3DEDGESTYLE, _
     isHidden As Boolean, _
     objectName As String) As Object
```

Creates and returns a **GwxArc** object with the specified attributes, in the current display. Angles are specified in degrees.

Note: (See also **GwxArc** and **GwxVisible**).
Available only in configure mode.

VBA example:

```
    'deselect button
    o.Selected = False
    ThisDisplay.DeleteObject ("TheArc")

    Dim obj As GwxArc
    Set obj = ThisDisplay.CreateArc( _
                        ArcPie, _
                        470, _
                        242, _
                        30, _
                        40, _
                        270, _
                        45, _
                        True, _
                        RGB(198, 255, 190), _
                        RGB(10, 100, 200), _
                        2, _
                        LineSolid, _
                        False, _
                        RGB(50, 50, 50), _
                        EdgeEtched, _
                        False, _
                        "TheArc")

    If obj Is Nothing Then
```

```
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    obj.UserDescription = "The Arc"
```

Function CreateBitmap

```
Function CreateBitmap(filename As String, _
     left As Single, _
     top As Single, _
     width As Single, _
     height As Single, _
     isTransparent As Boolean, _
     transparentColor As Long, _
     lineColor As Long, _
     lineWidth As Long, _
     lineStyle As GWXLINESTYLE, _
     hasShadow As Boolean, _
     shadowColor As Long, _
     edgeStyle As GWX3DEDGESTYLE, _
     isHidden As Boolean, _
     objectName As String) As Object
```

Creates and returns a **GwxBitmap** object with the specified attributes, in the current display. (See also **GwxVisible**). The **fileName** parameter refers to the file name of a BMP file.

Note: Available only in configure mode. See also **GwxVisible**.

VBA example:

```
    'deselect button
    o.Selected = False
    ThisDisplay.DeleteObject ("ThePicture")

    Dim obj As GwxBitmap

    Set obj = ThisDisplay.CreateBitmap( _
                        "meter.bmp", _
                        700, _
                        125, _
                        127, _
                        125, _
                        False, _
                        RGB(192, 192, 192), _
                        RGB(192, 192, 192), _
                        1, _
                        LineNone, _
                        False, _
                        RGB(192, 192, 192), _
                        EdgeEtched, _
                        False, _
                        "ThePicture")

    If obj Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
```

```
    End If

    obj.UserDescription = "The Picture"
```

Function CreateButton

```
Function CreateButton(buttonType As GWXBUTTONTYPE, _
     x As Single, _
     y As Single, _
     label As String, _
     alignment As GWXTEXTALIGNMENT, _
     stretchText As Boolean, _
     isFilled As Boolean, _
     fillColor As OLE_COLOR, _
     lineColor As OLE_COLOR, _
     lineWidth As Long, _lineStyle As Long, _
     hasShadow As Boolean, _
     shadowColor As OLE_COLOR, _
     edgeStyle As Long, _
     isHidden As Boolean, _
     objectName As String) As Object
```

Creates and returns a **GwxButton** object with the specified attributes, in the current display. A button object will be non-operational until a **GwxPick** object is attached to it.

Note: See also GwxButton, GwxPick, GwxDynamic, GwxText, and GwxVisible. *Available only in configure mode.*

Function CreateDigitalColorDynamic

```
Function CreateDigitalColorDynamic(visibleObjectName As String, _
     dynamicObjectName As String, _
     changeColorWhenTrue As Boolean, _
     changeFill As Boolean, _
     changeLine As Boolean, _
     changeShadow As Boolean, _
     fillColor As OLE_COLOR, _
     lineColor As OLE_COLOR, _
     shadowColor As OLE_COLOR, _
     dataSource As String) As Object
```

Creates a **GwxDigitalColor** object with the specified attributes and attaches it to the visible object with the specified object name. The object is created with one initial data connection. Additional data connections can be added by calling this function again for the same visible object. Returns the newly created **GwxDigitalColor** object on success, NULL if the operation failed.

Note: See also **GwxDigitalColor, GwxDigitalColorInfo,** and **GwxDynamic**). *Available only in configure mode.*

VBA example:

```
    Dim objRect As GwxRectangle

   'button
   o.Selected = False

    Call ThisDisplay.DeleteObject("TheRectangle")
    Set objRect = ThisDisplay.CreateRectangle( _
```

```
                                442, _
                                213, _
                                80, _
                                60, _
                                True, _
                                RGB(100, 200, 230), _
                                RGB(100, 200, 230), _
                                1, _
                                LineNone, _
                                False, _
                                RGB(192, 192, 192), _
                                EdgeEtched, _
                                False, _
                                "TheRectangle", _
                                True)
    If objRect Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    objRect.UserDescription = "The Rectangle"

    ThisDisplay.DeleteDynamic ("ObjDyn")
    Call ThisDisplay.CreateDigitalColorDynamic( _
                    "TheRectangle", _
                    "ObjDyn", _
                    True, _
                    True, _
                    False, _
                    False, _
                    RGB(213, 200, 200), _
                    RGB(213, 120, 120), _
                    RGB(213, 120, 120), _
                    "gfwsim.toggle.bool")
```

Function CreateDigitalSelector

```
Function CreateDigitalSelector(symbolName As String _
     ,dynamicObjectName As String) As Object
```

Creates a **GwxDigitalSelector** object with the specified attributes and attaches it to the **GwxSymbol** object with the specified object name. It returns the newly created **GwxDigitalSelector** object on success, NULL if the operation failed. The newly created object has no data connections; to add data connections, use **GwxDigitalSelector::SetConnectionInfo.**

Note: Available only in configure mode. See also **GwxDigitalSelector, GwxDigitalSelectorInfo, GwxSymbol, GwxVisible,** and **GwxDynamic**.

VBA example:

```
    o.Selected = False

    Dim objRect As GwxRectangle

    ThisDisplay.UngroupSymbol ("Symbol_0")

    Set objRect = ThisDisplay.GetVisibleObjectFromName( _
                            "Rectangle_0")
    objRect.Selected = True
```

```
    Dim objEll As GwxEllipse
    Set objEll = ThisDisplay.GetVisibleObjectFromName( _
                            "Ellipse_0")
    objEll.Selected = True

    Dim objCir As GwxEllipse
    Set objCir = ThisDisplay.GetVisibleObjectFromName( _
                            "Circle_0")
    objCir.Selected = True

    'Deselect the button:
    o.Selected = False

    'Use selected visibles to create a group:
    ThisDisplay.CreateSymbol ("Symbol_0")


    ThisDisplay.DeleteDynamic ("ObjectDyn")
    Dim objDyn As GwxDigitalSelector
    Set objDyn = ThisDisplay.CreateDigitalSelector( _
                            "Symbol_0", _
                            "ObjectDyn")

    If objRect Is Nothing Then
        MsgBox "No dynamic object has been created."
        Exit Sub
    Else
        MsgBox "A new dynamic object has been created."
    End If

    objDyn.UserDescription = "The Digital Selector"
    'Call objDyn.SetConnectionInfo(0, "gfwsim.sine.double", True)
    Call objDyn.SetConnectionInfo(1, "gfwsim.cosine.double", False)
    Call objDyn.SetConnectionInfo(2, "gfwsim.toggle.bool", True)
```

Function CreateEllipse

```
Function CreateEllipse(left As Single, _
     top As Single, _
     width As Single, _
     height As Single, _
     isFilled As Boolean, _
     fillColor As OLE_COLOR, _
     lineColor As OLE_COLOR, _
     lineWidth As Long, _
     lineStyle As GWXLINESTYLE, _
     hasShadow As Boolean, _
     shadowColor As OLE_COLOR, _
     edgeStyle As GWX3DEDGESTYLE, _
     isHidden As Boolean, _
     objectName As String) As Object
```

Creates and returns **GwxEllipse** object with the specified attributes, in the current display.

Note: (See also **GwxEllipse** and **GwxVisible**). *Available only in configure mode.*

VBA example:

```
    Dim objEll As GwxEllipse

    ThisDisplay.DeleteObject ("TheEllipse")
```

```
    'deselect button
    o.Selected = False
    Set objEll = ThisDisplay.CreateEllipse( _
                             442, _
                             230, _
                             80, _
                             30, _
                             True, _
                             RGB(123, 200, 123), _
                             RGB(123, 200, 123), _
                             1, _
                             LineNone, _
                             False, _
                             RGB(123, 200, 123), _
                             EdgeEtched, _
                             False, _
                             "TheEllipse")
    If objEll Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    objEll.UserDescription = "The Ellipse"
```

Function CreateFlashDynamic

```
Function CreateFlashDynamic(visibleObjectName As String, _
     dynamicObjectName As String, _
     hideObject As Boolean, _
     flashWhenTrue As Boolean, _
     altStateWhenOff As Boolean, _
     changeFill As Boolean, _
     changeLine As Boolean, _
     changeShadow As Boolean, _
     altFillColor As OLE_COLOR, _
     altLineColor As OLE_COLOR, _
     altShadowColor As OLE_COLOR) As Object
```

Creates a **GwxFlash** object with the specified attributes and attaches it to the visible object with the specified object name. Returns the newly created **GwxFlash** object on success, NULL if the operation failed.

Note: (See also **GwxFlash** and **GwxDynamic**). *Available only in configure mode.*

VBA example:

```
    Dim objRect As GwxRectangle

    'deselect button
    o.Selected = False

    Call ThisDisplay.DeleteObject("TheRectangle")
    Set objRect = ThisDisplay.CreateRectangle( _
                               442, _
                               213, _
                               80, _
                               60, _
                               True, _
                               RGB(100, 200, 230), _
                               RGB(100, 200, 230), _
```

```
                                1, _
                                LineNone, _
                                False, _
                                RGB(192, 192, 192), _
                                EdgeEtched, _
                                False, _
                                "TheRectangle", _
                                True)
    If objRect Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    objRect.UserDescription = "The Rectangle"

    ThisDisplay.DeleteDynamic ("TheFlashObject")
    Dim obj As GwxFlash

    Set obj = ThisDisplay.CreateFlashDynamic( _
                            "TheRectangle", _
                            "TheFlashObject", _
                            False, _
                            True, _
                            False, _
                            True, _
                            True, _
                            True, _
                            RGB(230, 230, 100), _
                            RGB(100, 100, 230), _
                            RGB(100, 100, 100))

    If obj Is Nothing Then
        MsgBox "No dynamic object has been created."
        Exit Sub
    Else
        MsgBox "A new dynamic object has been created."
    End If

    obj.UserDescription = "The Flash Object"
    obj.dataSource = "gfwsim.toggle.bool"
```

Function CreateHideDynamic

```
Function CreateHideDynamic(visibleObjectName As String, _
     dynamicObjectName As String, _
     hideWhenTrue As Boolean, _
     disableObject As Boolean) As Object
```

Creates a **GwxHide** object with the specified attributes and attaches it to the visible object with the specified object name. It returns the newly created **GwxHide** object on success, NULL if the operation failed.

Note: (See also **GwxHide** and **GwxDynamic**). *Available only in configure mode.*

VBA example:

```
    Dim objRect As GwxRectangle

    o.Selected = False

    Call ThisDisplay.DeleteObject("TheRectangle")
```

```
    Set objRect = ThisDisplay.CreateRectangle( _
                                442, _
                                213, _
                                80, _
                                60, _
                                True, _
                                RGB(100, 200, 230), _
                                RGB(100, 200, 230), _
                                1, _
                                LineNone, _
                                False, _
                                RGB(192, 192, 192), _
                                EdgeEtched, _
                                False, _
                                "TheRectangle", _
                                True)
    If objRect Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    objRect.UserDescription = "The Rectangle"

    ThisDisplay.DeleteDynamic ("TheHideObject")

    Dim obj As GwxHide
    Set obj = ThisDisplay.CreateHideDynamic( _
                          "TheRectangle", _
                          "TheHideObject", _
                          True, _
                          False)

    If obj Is Nothing Then
        MsgBox "No dynamic object has been created."
        Exit Sub
    Else
        MsgBox "A new dynamic object has been created."
    End If

    obj.UserDescription = "The Hide Object"
    obj.dataSource = "gfwsim.toggle.bool"
```

Sub CreateLayer

```
Sub CreateLayer(layerName As String)
```

Creates a new Layer in the display and assigns to it the name specified in the parameter.

VBA example:

```
    'If any exists...
    ThisDisplay.RemoveLayer ("Layer9999")

    ThisDisplay.CreateLayer ("Layer9999")
    MsgBox "A new layer has been created."

    MsgBox "On the Format menu, click Layers -> Remove" + _
           " to remove the layer Layer9999"
```

Function CreateLocationDynamic

```
Function CreateLocationDynamic(visibleObjectName As String, _
```

```
    dynamicObjectName As String, _
    offsetX As Single, _
    offsetY As Single, _
    slider As Boolean, _
    tracking As Boolean, _
    numberOfDetents As Integer, _
    continuousUpdate As Boolean) As Object
```

Creates a **GwxLocation** object with the specified attributes and attaches it to the visible object with the specified object name. The parameters **offsetX** and **offsetY** refer to the distance the object will travel from its current location. Returns the newly created **GwxLocation** object on success, NULL if the operation failed.

Note: (See also **GwxLocation** and **GwxDynamic**). *Available only in configure mode.*

VBA example:

```
Dim objRect As GwxRectangle

o.Selected = False

Call ThisDisplay.DeleteObject("TheRectangle")
Set objRect = ThisDisplay.CreateRectangle( _
                            442, _
                            213, _
                            80, _
                            60, _
                            True, _
                            RGB(100, 200, 230), _
                            RGB(100, 200, 230), _
                            1, _
                            LineNone, _
                            False, _
                            RGB(192, 192, 192), _
                            EdgeEtched, _
                            False, _
                            "TheRectangle", _
                            True)
If objRect Is Nothing Then
    MsgBox "Operation failed."
    Exit Sub
Else
    MsgBox "A new object has been created."
End If

objRect.UserDescription = "The Rectangle"

'If any exists ...
ThisDisplay.DeleteDynamic ("TheLocationDynamicObject")

Dim obj As GwxLocation
Set obj = ThisDisplay.CreateLocationDynamic( _
                      "TheRectangle", _
                      "TheLocationDynamicObject", _
                      50, _
                      50, _
                      True, _
                      True, _
                      0, _
                      True)

If obj Is Nothing Then
```

```
        MsgBox "No dynamic object has been created."
        Exit Sub
    Else
        MsgBox "A new dynamic object has been created."
    End If

    obj.UserDescription = "The Location Dynamic Object"
    obj.dataSource = "gfwsim.cosine.double"
```

Function CreateMetafile

```
Function CreateMetafile(filename As String, _
     left As Single, _
     top As Single, _
     width As Single, _
     height As Single, _
     objectName As String) As Object
```

Creates and returns a **GwxMetafile** object with the specified attributes, in the current display. (See also **GwxVisible**). The **fileName** parameter refers to the file name of a WMF/EMF file.

Note: Available only in configure mode. See also **GwxVisible**.

VBA example:

```
    o.Selected = False
    ThisDisplay.DeleteObject ("ThePicture")

    Dim objPic As GwxMetafile
    Set objPic = ThisDisplay.CreateMetafile( _
                            "valve.emf", _
                            700, _
                            125, _
                            90, _
                            120, _
                            "ThePicture")
    If objPic Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    objPic.UserDescription = "The Picture"
```

Function CreatePickDynamic

```
Function CreatePickDynamic(visibleObjectName As String, _
     dynamicObjectName As String, _
     pickAction As GWXPICKACTION, _
     pickType As GWXBUTTONTYPE, _
     executionTrigger As GWXEXECUTIONTRIGGER, _
     mouseButton As GWXMOUSEBUTTON, _
     initiallySelected As Boolean, _
     groupName As String, _
     filename As String, _
     modal As Boolean, _
     center As Boolean, _
     value1 As String, _
     value2 As String, _
```

```
    value3 As String) As Object
```

Creates a **GwxPick** object with the specified attributes and attaches it to the visible object with the specified object name. Valid values for **GWXPICKACTION** are:

PickLoadDisplay = 0
PickDragDropLoad = 1
PickPopupWindow = 2
PickDownloadValue = 3
PickToggleValue = 4
PickLaunchApp = 5
PickClose = 6
PickRunScript = 7
PickCustomCommand = 8
PickEmbeddedWindow = 9
PickDisplayBack = 10
PickDisplayForward = 11
PickSetAliases = 12
PickAliasesDialog = 13
PickLayerHideOn = 14
PickLayerHideOff = 15
PickLayerHideToggle = 16
PickPopupUserMenu = 17
PickSwitchLanguage = 18
PickSelectThema = 19
PickLaunchWebPage = 20

Valid values for **GWXBUTTONTYPE** are:

ButtonNormal = 0
ButtonCheck = 1
ButtonRadio = 2

Valid values for **GWXEXECUTIONTRIGGER** are:

TriggerOnDown = 1
TriggerWhileDown = 2
TriggerOnDnWhileDn = 3
TriggerOnUp = 4
TriggerOnDnOnUp = 5
TriggerWhileDnOnUp = 6
TriggerOnDnWhileDnOnUp = 7

Valid values for **GWXMOUSEBUTTON** are:

MouseButtonLeft = 0
MouseButtonMiddle = 1
MouseButtonRight = 2

Depending upon the value of **pickAction**, some parameters may be ignored by GraphWorX32. Returns the newly created **GwxPick** object on success, NULL if the operation failed.

Note: Available only in configure mode.
(See also **GwxPick** and **GwxDynamic**).

VBA example:

```
    o.Selected = False
```

```
    ThisDisplay.DeleteObject ("Button_0")

    Dim obj As GwxButton
    Set obj = ThisDisplay.CreateButton( _
                        ButtonNormal, _
                        440, _
                        250, _
                        "The Button", _
                        TextAlignCenter, _
                        False, _
                        True, _
                        RGB(120, 220, 120), _
                        RGB(0, 0, 120), _
                        1, _
                        EdgeNone, _
                        False, _
                        RGB(100, 100, 100), _
                        EdgeNone, _
                        False, _
                        "Button_0")

    If obj Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    obj.UserDescription = "The Button"

    Call ThisDisplay.CreatePickDynamic("Button_0", _
                                  "PickDyn_0", _
                                  PickRunScript, _
                                  ButtonNormal, _
                                  TriggerWhileDown, _
                                  MouseButtonLeft, _
                                  False, _
                                  "", _
                                  "", _
                                  False, _
                                  True, _
                                  "", _
                                  "", _
                                  "")
```

Function CreatePolyline

```
Function CreatePolyline(vertices, _
     isFilled As Boolean, _
     fillColor As OLE_COLOR, _
     lineColor As OLE_COLOR, _
     lineWidth As Long, _
     lineStyle As GWXLINESTYLE, _
     hasShadow As Boolean, _
     shadowColor As OLE_COLOR, _
     edgeStyle As GWX3DEDGESTYLE, _
     isHidden As Boolean, _
     objectName As String) As Object
```

Creates and returns a **GwxLine** object with the specified attributes, in the current display. The parameter **vertices** is an array of float values such that the elements 0, 2, 4, 6… of the array

are x-coordinates of the vertices of the line, and elements 1, 3, 5, 7… of the array are y-coordinates of the vertices of the line.

Note: See also **GwxLine** and **GwxVisible**. When calling this method from a C++ application, the **vertices** parameter should be a "safe array" with the lower bound set to 1. *Available only in configure mode.*

VBA example:

```
    'If any exists
ThisDisplay.DeleteObject ("ThePolyLineObject")
o.Selected = False
Dim objPol As GwxLine

Dim lVertices(8) As Double

lVertices(1) = 683     'x
    lVertices(2) = 149 'y
lVertices(3) = 911     'x
    lVertices(4) = 239 'y
lVertices(5) = 708     'x
    lVertices(6) = 269 'y
lVertices(7) = 876     'x
    lVertices(8) = 274 'y


Set objPol = ThisDisplay.CreatePolyline( _
                        lVertices, _
                        True, _
                        RGB(100, 200, 255), _
                        RGB(100, 200, 255), _
                        1, _
                        LineNone, _
                        False, _
                        RGB(0, 0, 0), _
                        EdgeEtched, _
                        False, _
                        "ThePolyLineObject")

If objPol Is Nothing Then
    MsgBox "Operation failed."
    Exit Sub
Else
    MsgBox "A new object has been created."
End If

objPol.UserDescription = "The PolyLine"
```

Function CreateProcessPoint

```
Function CreateProcessPoint(textObjectName As String, _
     dynamicObjectName As String, _
     dataType As GWXDATATYPE, _
     update As Boolean, _
     dataEntry As Boolean, _
     hasInitialValue As Boolean, _
     initialValue, _
     format As String) As Object
```

Creates a **GwxProcessPoint** object with the specified attributes and attaches it to the **GwxText** object with the specified object name. Returns the newly created **GwxProcessPoint** object on success, NULL if the operation failed.

Note: See also **GwxProcessPoint, GwxText, GwxVisible,** and **GwxDynamic**. *Available only in configure mode.*

VBA example:

```
o.Selected = False

ThisDisplay.DeleteObject ("TheText")


Dim objText As GwxText
Set objText = ThisDisplay.CreateText( _
                          448, _
                          232, _
                          "The Text", _
                          TextAlignCenter, _
                          False, _
                          True, _
                          RGB(200, 200, 230), _
                          RGB(10, 10, 50), _
                          1, _
                          LineNone, _
                          False, _
                          RGB(0, 0, 0), _
                          EdgeEtched, _
                          False, _
                          "TheText")
If objText Is Nothing Then
    MsgBox "Operation failed."
    Exit Sub
Else
    MsgBox "A new text object has been created."
End If

objText.UserDescription = "The Text"



ThisDisplay.DeleteDynamic ("TheProcessPointObject")

Dim obj As GwxProcessPoint
Set obj = ThisDisplay.CreateProcessPoint( _
                      "TheText", _
                      "TheProcessPointObject", _
                      DataTypeDouble, _
                      True, _
                      False, _
                      True, _
                      0, _
                      "x.xe")

ThisDisplay.RefreshWindow

If obj Is Nothing Then
    MsgBox "No dynamic object has been created."
    Exit Sub
Else
    MsgBox "A new dynamic object has been created."
End If

obj.UserDescription = "The Process Point"
obj.dataSource = "gfwsim.cosine.double"
```

Function CreateRectangle

```
Function CreateRectangle(left As Single, _
     top As Single, _
     width As Single, _
     height As Single, _
     isFilled As Boolean, _
     fillColor As OLE_COLOR, _
     lineColor As OLE_COLOR, _
     lineWidth As Long, _
     lineStyle As GWXLINESTYL
     hasShadow As Boolean, _
     shadowColor As OLE_COLOR, _
     edgeStyle As GWX3DEDGESTYLE, _
     isHidden As Boolean, _
     objectName As String, _
     rounded As Boolean) As Object
```

Creates and returns a **GwxRectangle** object with the specified attributes, in the current display.

Note: (See also **GwxRectangle** and **GwxVisible**). *Available only in configure mode.*

VBA example:

```
Dim objRect As GwxRectangle

o

Call ThisDisplay.DeleteObject("TheRectangle")
Set objRect = ThisDisplay.CreateRec
                          442, _
                          213, _
                          80, _
                          60, _
                          True, _
                          RGB(100, 200, 230), _
                          RGB(1
                          1, _
                          LineNone,
                          False, _
                          RGB(192, 192,
                          EdgeEtche
                          False, _
                          "TheRe

If objRect Is Nothing Then
    MsgBox "O
    E
Else
    Msg
E

objRect.UserDescription = "The Rectangle"
```

Function CreateRotationDynamic

```
Function CreateRotationDynamic(visibleObjectName As String, _
     dynamicObjectName As String, _
     startAngle As Single, _
     endAngle As Single, _
```

```
    pivotX As Single,
    pivotY As Single, _
    clockwise As Boolean, _
    dial As Boolean, _
    tracking As Boolean, _
    numberOfDetents As Integer, _
    continuousUpdate As Boolean) As Object
```

Creates a **GwxRotation** object with the specified attributes and attaches it to the visible object with the specified object name. The parameters **pivotX** and **pivotY** are offsets from the center of the object. Angles are specified in degrees. Returns the newly created **GwxRotation** object on success, NULL if the operation failed.

Note: (See also **GwxRotation** and **GwxDynamic**). *Available only in configure mode.*

VBA example:

```
Dim objRect As GwxRectangle

o.Selected = False

Call ThisDisplay.DeleteObject("TheRectangle")

                            442, _
                            213, _
                            80, _
                            60, _
                            True, _
                            RGB(100, 200, 230), _
                            RGB(100, 200, 230), _
                            1, _
                            LineNone,
                            False, _
                            RGB(192, 192, 192), _
                            EdgeEtched, _
                            False, _
                            "TheRe


    MsgBox "Operation failed."
    Exit Sub
Else
    Msg


objRect.UserDescription = "The Rectangle"

T

Dim obj As GwxRotation
Set obj = ThisDisplay.CreateRotationDynamic( _
                        "TheRectangle", _
                        "TheRo

                        270, _
                        20, _
                        30, _
                        True, _
                        True, _
                        True, _
                        0, _
                        True)
```

```
If obj Is Nothing Then
    MsgBox "No dynamic object has been created."
    Exit Sub
Else
    MsgBox "A new dynamic object has been created."
End If

obj.UserDescription = "The Rotation Dynamic Object"
obj.dataSource = "gfwsim.cosine.double"
'obj.RangeOverride = True

'Dim objPoint As GwxPoint
'Set objPoint = obj.GetDataSourcePointObject
'objPoint.highRange = 1
'objPoint.lowRange = -1
```

Function CreateSizeDynamic

```
Function CreateSizeDynamic(visibleObjectName As String, _
     dynamicObjectName As String, _
     sizeType As GWXSIZEDYNTYPE, _
     clip As Boolean, _
     startSize As Single, _
     endSize As Single) As Object
```

Creates a **GwxSize** object with the specified attributes and attaches it to the visible object with the specified object name. The parameters **startSize** and **endSize** are percentage values in the range of 0.0 to 1.0. Valid values for **GWXSIZEDYNTYPE** are:

SizeLeft = 0
SizeRight = 1
SizeUp = 2
SizeDown = 3
SizeUpLeft = 4
SizeUpRight = 5
SizeDownLeft = 6
SizeDownRight = 7
SizeLeftRight = 8
SizeUpDown = 9
SizeLeftRightBias = 10
SizeUpDownBias = 11
SizeAllFour = 12
SizeLeftRightUp = 13
SizeLeftRightDown = 14
SizeUpDownLeft = 15
SizeUpDownRight = 16

Function returns the newly created **GwxSize** object on success, NULL if the operation failed.

Note: (See also **GwxSize** and **GwxDynamic**). *Available only in configure mode.*

VBA example:

```
Dim objRect As GwxRectangle

o.Selected = False

Call ThisDisplay.DeleteObject("TheRectangle")
Set objRect = ThisDisplay.CreateRectangle( _
                            442, _
                            213, _
```

```
                                80, _
                                60, _
                                True, _
                                RGB(100, 200, 230), _
                                RGB(100, 200, 230), _
                                1, _
                                LineNone, _
                                False, _
                                RGB(192, 192, 192), _
                                EdgeEtched, _
                                False, _
                                "TheRectangle", _
                                True)
    If objRect Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new object has been created."
    End If

    objRect.UserDescription = "The Rectangle"

    ThisDisplay.DeleteDynamic ("TheSizeDynamic")

    Dim obj As GwxSize
    Set obj = ThisDisplay.CreateSizeDynamic( _
                           "TheRectangle", _
                           "TheSizeDynamic", _
                           SizeUpDown, _
                           True, _
                           -1, _
                           1)

    If obj Is Nothing Then
        MsgBox "No dynamic object has been created."
        Exit Sub
    Else
        MsgBox "A new dynamic object has been created."
    End If

    obj.UserDescription = "The Rotation Dynamic Object"
    obj.dataSource = "gfwsim.cosine.double"
```

Function CreateStateField

```
Function CreateStateField(textObjectName As String, _
     dynamicObjectName As String, _
     update As Boolean, _
     dataEntry As Boolean, _
     hasInitialValue As Boolean, _
     initialValue, _
     stateConfiguration As String, _
     defaultState As String) As Object
```

The **stateConfiguration** parameter has the following format:
value1<TAB>statestring1<CR-LF>value2<TAB>statestring2<CR-LF> … etc.
The **defaultState** parameter is the string shown when no matching state is found.
Note: For more information, see **CreateProcessPoint()**. *Available only in configure mode.*

VBA example:

```
    o.Selected = False
```

```
'if any exists...

ThisDisplay.DeleteObject ("TheText")


Dim objText As GwxText
Set objText = ThisDisplay.CreateText( _
                          448, _
                          232, _
                          "The Text", _
                          TextAlignCenter, _
                          False, _
                          True, _
                          RGB(200, 200, 230), _
                          RGB(10, 10, 50), _
                          1, _
                          LineNone, _
                          False, _
                          RGB(0, 0, 0), _
                          EdgeEtched, _
                          False, _
                          "TheText")
If objText Is Nothing Then
    MsgBox "Operation failed."
    Exit Sub
Else
    MsgBox "A new text object has been created."
End If

objText.UserDescription = "The Text"


ThisDisplay.DeleteDynamic ("TheStateField")

Dim obj As Object

Dim strStateConfiguration As String

strStateConfiguration = _
        "0" + Chr(9) + "Line1" + Chr(13) + _
        "1" + Chr(9) + "Line2"

Set obj = ThisDisplay.CreateStateField( _
                    "TheText", _
                    "TheStateField", _
                    True, _
                    False, _
                    True, _
                    0, _
                    strStateConfiguration, _
                    "<#Building#>")
If obj Is Nothing Then
    MsgBox "No dynamic object has been created."
    Exit Sub
Else
    MsgBox "A new dynamic object has been created."
End If


obj.dataSource = "gfwsim.toggle.bool"
```

Function CreateSymbol

```
Function CreateSymbol(objectName As String) As Object
```

Groups all currently selected objects into a **GwxSymbol** object, giving the resulting symbol object the specified object name. It returns the resulting **GwxSymbol** object.

Note: See also **GwxSymbol** and **GwxVisible**. Available only in configure mode.

VBA example:

```
    o.Selected = False

    Dim objRect As GwxRectangle

    ThisDisplay.UngroupSymbol ("Symbol_0")
    ThisDisplay.DeselectAllObjects

    Set objRect = ThisDisplay.GetVisibleObjectFromName( _
                            "Rectangle_0")
    MsgBox "Select the Rectangle_0."
    objRect.Selected = True


    Dim objEll As GwxEllipse
    Set objEll = ThisDisplay.GetVisibleObjectFromName( _
                            "Ellipse_0")
    MsgBox "Select the Ellipse_0."
    objEll.Selected = True

    Dim objCir As GwxEllipse
    Set objCir = ThisDisplay.GetVisibleObjectFromName( _
                            "Circle_0")
    MsgBox "Select the Circle_0."
    objCir.Selected = True

    'Deselect the button:
    o.Selected = False

    MsgBox "And end up create a symbol."
    'Use selected visibles to create a group:
    ThisDisplay.CreateSymbol ("Symbol_0")
```

Function CreateText

```
Function CreateText(x As Single, _
     y As Single, _
     text As String, _
     alignment As GWXTEXTALIGNMENT, _
     stretchText As Boolean, _
     isFilled As Boolean, _
     fillColor As OLE_COLOR, _
     lineColor As OLE_COLOR, _
     lineWidth As Long, _
     lineStyle As GWXLINESTYLE, _
     hasShadow As Boolean, _
     shadowColor As OLE_COLOR, _
     edgeStyle As GWX3DEDGESTYLE, _
     isHidden As Boolean, _
     objectName As String) As Object
```

Creates and returns a **GwxText** object with the specified attributes, in the current display.

Note: See also **GwxText** and **GwxVisible**. Available only in configure mode.

VBA example:

```
    o.Selected = False
    'if any exists...

    ThisDisplay.DeleteObject ("TheText")


    Dim objText As GwxText
    Set objText = ThisDisplay.CreateText( _
                              448, _
                              232, _
                              "The Text", _
                              TextAlignCenter, _
                              False, _
                              True, _
                              RGB(200, 200, 230), _
                              RGB(10, 10, 50), _
                              1, _
                              LineNone, _
                              False, _
                              RGB(0, 0, 0), _
                              EdgeEtched, _
                              False, _
                              "TheText")
    If objText Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new text object has been created."
    End If

    objText.UserDescription = "The Text"
```

Function CreateTimedate

```
Function CreateTimedate(textObjectName As String, _
     dynamicObjectName As String, _
     formatType As GWXTIMEDATEFORMATTYPE, _
     timeFormat As String, _
     dateFormat As String) As Object
```

Creates a **GwxTimedate** object with the specified attributes and attaches it to the **GwxText** object with the specified object name. Returns the newly created **GwxTimedate** object on success, NULL if the operation failed.

Note: See also **GwxTimedate, GwxText, GwxVisible,** and **GwxDynamic**. Available only in configure mode.

GWXTIMEDATEFORMATTYPE:
TimeOnly = 0,
DateOnly = 1,
TimeAndDate = 2

VBA example:

```
    o.Selected = False
    'if any exists...

    ThisDisplay.DeleteObject ("TheText")
```

```
    Dim objText As GwxText
    Set objText = ThisDisplay.CreateText( _
                              448, _
                              232, _
                              "The Text", _
                              TextAlignCenter, _
                              False, _
                              True, _
                              RGB(200, 200, 230), _
                              RGB(10, 10, 50), _
                              1, _
                              LineNone, _
                              False, _
                              RGB(0, 0, 0), _
                              EdgeEtched, _
                              False, _
                              "TheText")
    If objText Is Nothing Then
        MsgBox "Operation failed."
        Exit Sub
    Else
        MsgBox "A new text object has been created."
    End If

    objText.UserDescription = "The Text"

    ThisDisplay.DeleteDynamic ("TheTimeAndDateObject")

    Dim obj As GwxTimedate

    Set obj = ThisDisplay.CreateTimedate( _
                            "TheText", _
                            "TheTimeAndDateObject", _
                            TimeAndDate, _
                            "h:mm:ss tt", _
                            "M/d/yyyy")
    If obj Is Nothing Then
        MsgBox "No dynamic object has been created."
        Exit Sub
    Else
        MsgBox "A new dynamic object has been created."
    End If
```

Function DeleteDynamic

```
Function DeleteDynamic(objectName As String) As Boolean
```

Deletes the dynamic object with the specified object name. Returns TRUE for success, FALSE for failure.

Available only in configure mode.

VBA example:

```
    ThisDisplay.DeleteDynamic ("Rectangle2")
```

Function DeleteObject

```
Function DeleteObject(objectName As String) As Boolean
```

Deletes the visible object with the specified object name. Returns TRUE for success, FALSE for failure.

Note: Available only in configure mode.

VBA example:

```
ThisDisplay.DeleteObject ("Rectangle2")
```

Sub DeselectAllObjects

```
Sub DeselectAllObjects()
```

Deselects all currently selected visible objects.

Note: Available only in configure mode.

VBA example:

```
Dim obj As Object
Dim obj2 As Object

'Select rectangles:
Set obj = ThisDisplay.GetVisibleObjectFromName("Rectangle")
Set obj2 = ThisDisplay.GetVisibleObjectFromName("Rectangle2")

MsgBox "Now we will select the rectangles."
obj.Selected = True
obj2.Selected = True

MsgBox "Now we are going to deselect both rectangles."

ThisDisplay.DeselectAllObjects
```

Function DisplayBack

```
Function DisplayBack() As Boolean
```

Opens the previously loaded display in the display file history.

Note: (Not available in ActiveX ToolWorX).

VBA example:

```
ThisDisplay.DisplayBack
```

Function DisplayForward

```
Function DisplayForward() As Boolean
```

Opens the next display in the display file history.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
ThisDisplay.DisplayForward
```

VB example:

```
ThisDisplay.DisplayForward
```

Sub DuplicateSelection

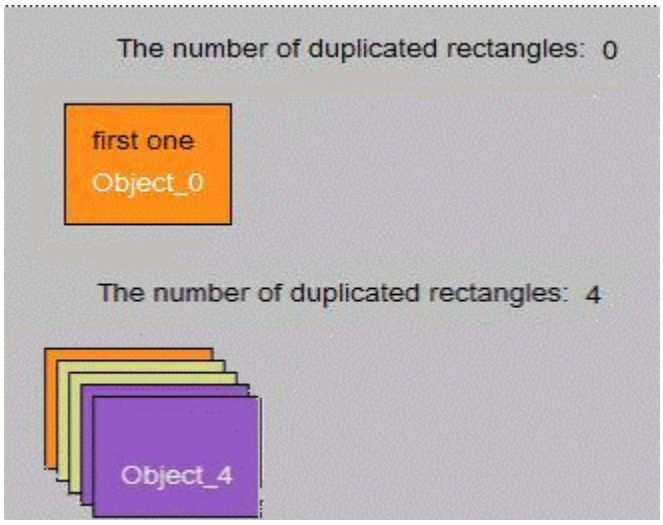
```
Sub DuplicateSelection()
```

It creates duplicates of the currently selected objects.

Note: Available only in configure mode. Not available for displays running in GWXview32 objects.

VBA example:

```
ThisDisplay.DuplicateSelection
```



VBA example:

```
'We assume we have created a rectangle (Object_X)
'with a text object (rectText) to watch how many
'rectangles is duplicated.
'"Counter" is a helper variable to indicate
'the number of the newly created rectangles.

'Helper declarations:
Dim x As Single
Dim y As Single
Dim width As Single
Dim height As Single

Dim obj As Object
'Get the helper index (Counter) to select
'the right rectangle.
Set obj = ThisDisplay.GetVisibleObjectFromName("Counter")
Dim txtCounter As GwxText 'intellisense
Set txtCounter = obj
Set obj = ThisDisplay.GetVisibleObjectFromName( _
    "Object_" + CStr(txtCounter.text))

Dim rect As GwxRectangle
Set rect = obj 'intellisense

rect.Selected = True
o.Selected = False 'Do not duplicate the button selection.

'Duplicate currantly selected object.
ThisDisplay.DuplicateSelection

'increase the number of rectangles
txtCounter.text = CStr(Val(txtCounter.text) + 1)

'Object_0 is automatically copied to a new object with
'the name Object_0,1,2...
Set obj = ThisDisplay.GetVisibleObjectFromName( _
    "Object_" + CStr(txtCounter.text))
```

```
    Set rect = obj
    Call rect.GetObjectDimensions(x, y, width, height)
    Call rect.SetObjectDimensions(x + 6, y + 6, width, height)

    'Set the random color:
    Dim a As Long
    Randomize (Second(Time) + Minute(Time) * 60 + Hour(Time) * 3600)
    rect.fillColor = RGB(Int(255 * Rnd), _
                         Int(255 * Rnd), _
                         Int(255 * Rnd))
    Dim txt As GwxText
    Set txt = ThisDisplay.GetVisibleObjectFromName("rectText")


    Call txt.GetObjectDimensions(x, y, width, height)
    'Shift the label by 6 pixels.
    Call txt.SetObjectDimensions(x + 6, y + 6, width, height)
    txt.Selected = True
    txt.text = "Object_" + CStr(txtCounter.text)
    rect.Selected = False
    ThisDisplay.MoveSelectionToFront

    If Val(txtCounter.text) > 10 Then
        Dim j As Long
        For j = 1 To 11
            ThisDisplay.DeleteObject ("Object_" + CStr(j))
        Next j
        'Set the former text and position of the Object rectText
        Call txt.SetObjectDimensions(x - 6 * 10, y - 6 * 10, width, height)
        txt.objectName = "rectText"
        txt.text = "Object_0"
        txtCounter.text = "0"
    End If

    txt.Selected = False
```

Sub EnableRuntimeEvents

```
Sub EnableRuntimeEvents(LeftButton As Boolean, _
     MiddleButton As Boolean, _
     RightButton As Boolean, _
     MouseMove As Boolean, _
     Keyboard As Boolean)
```

This method enables or disables firing of keyboard and mouse events into VBA.

LeftButton

Set it to True to get DisplayMouse events related to the left button actions.

MiddleButton

Set it to True to get DisplayMouse events related to the middle button actions.

RightButton

Set it to True to get DisplayMouse events related to the right button actions.

MouseMove

Set it to True to get DisplayMouseMove events.

Keyboard

Set it to True to get Keyboard events.

VBA example:

```
    If ThisDisplay.IsEnabledRuntimeZoomAndPan Then
        ThisDisplay.EnableRuntimeZoomAndPan (False)
    Else
        ThisDisplay.EnableRuntimeZoomAndPan (True)
    End If
```

VBA example:

```
Private Sub GwxDisplay_PreRuntimeStart()

    ' Enable right click events only
    Call ThisDisplay.EnableRuntimeEvents( _
      False, False, True, False, False)

End Sub
```

VB example:

```
    If ThisDisplay.IsEnabledRuntimeZoomAndPan Then
        ThisDisplay.EnableRuntimeZoomAndPan (False)
    Else
        ThisDisplay.EnableRuntimeZoomAndPan (True)
    End If
```

Sub ExitApplication

```
Sub ExitApplication()
```

Exits GraphWorX32.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
ThisDisplay.ExitApplication
```

VB example:

```
ThisDisplay.ExitApplication
```

Function ExportBitmapCaptureOfView

```
Function ExportBitmapCaptureOfView(filename As String) As Boolean
```

Exports a bitmap (BMP) file with specified name. The bitmap is an image of the current GraphWorX32 view.

Note: Not available for displays running in GWXview32 objects.

VBA exmple:

```
    Dim strCurrDir As String
    strCurrDir = ThisDisplay.GetFilePath

    Dim strFileName As String
    strFileName = "screenshot.bmp"
```

```
'Add a backslash at the end.
strFileName = strCurrDir + "\" + strFileName
ThisDisplay.ExportBitmapCaptureOfView (strFileName)
```

VB example:

```
Dim strCurrDir
strCurrDir = ThisDisplay.GetFilePath

Dim strFileName
strFileName = "screenshot.bmp"

'Add a backslash at the end.
strFileName = strCurrDir + "\" + strFileName
ThisDisplay.ExportBitmapCaptureOfView (strFileName)
```

Function FileNew

```
Function FileNew() As Boolean
```

Function starts a new (empty) display. Returns TRUE for success, FALSE for failure.

It resets the settings from the previous display.

VBA example:

```
'Save the current display.
ThisDisplay.FileSave

'Starts a new empty display.
ThisDisplay.FileNew
```

VB example:

```
'Save the current display.
ThisDisplay.FileSave

'Starts a new empty display.
ThisDisplay.FileNew
```

Function FileOpen

```
Function FileOpen(filename As String) As Boolean
```

Method loads the specified display file. Returns TRUE for success, FALSE for failure.

Note: In ActiveX ToolWorX this method only opens resource displays, not disk file displays.

Note: You should validate the result of this function before running additional code, and should avoid sending bursts of display load requests. A new display load request might arrive before the current display load request is completed.

VBA example:

```
'Save the current display.
ThisDisplay.FileSave

MsgBox "Open the display ""EmbeddedWindow.gdf""."

'Open a display.
ThisDisplay.FileOpen ("EmbeddedWindow.gdf")
```

VB example:

```
'Save the current display.
ThisDisplay.FileSave

MsgBox "Open the display ""EmbeddedWindow.gdf""."

'Open a display.
ThisDisplay.FileOpen ("EmbeddedWindow.gdf")
```

Function FileOpenSetAliases

```
Function FileOpenSetAliases(filename As String, _
     aliasCommandType As GWXALIASCOMMAND, _
     aliasCommandString As String) As Boolean
```

Opens the specified file and initially sets the specified aliases. If currently in runtime mode, the aliases will be set before any tags are requested. The parameter **aliasCommandType** can be one of the following values:

AliasNoCommand = 0
AliasSetFromFile = 1
AliasSetDirect = 2

If **aliasCommandType** is **AliasSetFromFile**, **aliasCommandString** should be a filename that specifies a text file containing tab-separated pairs of alias names and alias definitions (see also **SetAliasesFromFile()**). If **aliasCommandType** is **AliasSetDirect**, **aliasCommandString** should have the following format:

aliasName1<TAB>aliasDef1<CR-LF>aliasName2<TAB>aliasDef2<CR-LF> … etc.

Note: In ActiveX ToolWorX this method only opens resource displays, not disk file displays.

VBA example:

```
'There are assumed 3 process points
'in the popup display to run correctly this
'example.
'Put this code (optional) in "DataSource" in the
'Property Inspector in each of the process points:
'ProcessPont N.1:
    'x= $"SourceAlias1: The value is "$ + ...
    '... tostring(<<SourceAlias1>>) + $" units."$
'ProcessPont N.2:
    'x= $"SourceAlias2: The value is "$ + ...
    '... tostring(<<SourceAlias2>>) + $" units."$
'ProcessPont N.3:
    'x= $"SourceAlias3: The value is "$ + ...
    '... tostring(<<SourceAlias3>>) + $" units."$
'Adjust properties according to the string lenght.
'See "Creating Expressions Using the Expression Editor"
'in the help.

'Open a popup window:
Dim strFileName As String
strFileName = "DisplayWithAliases.gdf"

'Save your display before you launch the new one.
'It is not necessary.
ThisDisplay.FileSave

'Compose a command string:
Dim strAliasCommand As String
```

```
    strAliasCommand = "SourceAlias1" + Chr(9) + _
                        "100+abs({{gfwsim.sine.double}})*500" + _
                      Chr(13) + _
                      "SourceAlias2" + Chr(9) + _
                        "abs({{gfwsim.sine.double}})*1000" + _
                      Chr(13) + _
                      "SourceAlias3" + Chr(9) + _
                        "120+5*{{gfwsim.sine.double}}"
    'Execute the function:
    Call ThisDisplay.FileOpenSetAliases(strFileName, _
                                        AliasSetDirect, _
                                        strAliasCommand)
    '... or you can use a file, where the same
    'commands are written. For the file applyFor the tab spaces :
    'Call ThisDisplay.FileOpenSetAliases(strFileName, _
                                        AliasSetFromFile, _
                                        "AliasCommandString.txt")
    'Assume we have this file in the same directory.
```

VB example:

```
    'There are assumed 3 process points
    'in the popup display to run correctly this
    'example.
    'Put this code (optional) in "DataSource" in the
    'Property Inspector in each of the process points:
    'ProcessPont N.1:
        'x= $"SourceAlias1: The value is "$ + ...
        '... tostring(<<SourceAlias1>>) + $" units."$
    'ProcessPont N.2:
        'x= $"SourceAlias2: The value is "$ + ...
        '... tostring(<<SourceAlias2>>) + $" units."$
    'ProcessPont N.3:
        'x= $"SourceAlias3: The value is "$ + ...
        '... tostring(<<SourceAlias3>>) + $" units."$
    'Adjust properties according to the string lenght.
    'See "Creating Expressions Using the Expression Editor"
    'in the help.

    'Open a popup window:
    Dim strFileName
    strFileName = "DisplayWithAliases.gdf"

    'Save your display before you launch the new one.
    'It is not necessary.
    ThisDisplay.FileSave

    'Compose a command string:
    Dim strAliasCommand
    strAliasCommand = "SourceAlias1" + Chr(9) + _
                        "100+abs({{gfwsim.sine.double}})*500" + _
                      Chr(13) + _
                      "SourceAlias2" + Chr(9) + _
                        "abs({{gfwsim.sine.double}})*1000" + _
                      Chr(13) + _
                      "SourceAlias3" + Chr(9) + _
                        "120+5*{{gfwsim.sine.double}}"
    'Execute the function:
    Call ThisDisplay.FileOpenSetAliases(strFileName, _
                                        AliasSetDirect, _
                                        strAliasCommand)
    '... or you can use a file, where the same
    'commands are written. For the file applyFor the tab spaces :
```

```
    'Call ThisDisplay.FileOpenSetAliases(strFileName, _
    '                                    AliasSetFromFile, _
    '                                   "AliasCommandString.txt")
    'Assume we have this file in the same directory.
```

Function FilePrint

```
Function FilePrint() As Boolean
```

Prints the current display. Returns TRUE for success, FALSE for failure.

Note: (Not available in ActiveX ToolWorX).

VBA example:

```
    'Prints the current display.
    ThisDisplay.FilePrint
```

VB example:

```
    'Prints the current display.
    ThisDisplay.FilePrint
```

Function FileSave

```
Function FileSave() As Boolean
```

Function saves the current display. Returns TRUE for success, FALSE for failure.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
    ThisDisplay.FileSave
```

VB example:

```
    ThisDisplay.FileSave
```

Function FileSaveAs

```
Function FileSaveAs(filename As String) As Boolean
```

Saves the current display using the specified file name. Returns TRUE for success, FALSE for failure.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
     Dim fileName As String
    fileName = ThisDisplay.GetFileName

    If ThisDisplay.FileSaveAs(fileName) Then
        MsgBox "The file has been successfully saved."
    End If
```

VB example:

```
    Dim fileName
    fileName = ThisDisplay.GetFileName

    If ThisDisplay.FileSaveAs(fileName) Then
        MsgBox "The file has been successfully saved."
```

```
    End If
```

Function FileSaveAsType

```
Function FileSaveAsType(filename As String, _
     displayType As Integer) As Boolean
```

Saves the current display using the specified file name. The value of displayType should be zero for a VBA-display or one for a non-VBA-display. Returns TRUE for success, FALSE for failure.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
    'Saves the current file with the
    'current name.

    Dim VBAflag As Byte
    'Does this display contain a VBA part?
    If ThisDisplay.GetFileContainsVBA Then
        VBAflag = 0
    Else
        VBAflag = 1
    End If

    Call ThisDisplay.FileSaveAsType( _
        ThisDisplay.GetFileName, _
        VBAflag)
```

VB example:

```
   'Saves the current file with the
    'current name.

    Dim VBAflag
    'Does this display contain a VBA part?
    If ThisDisplay.GetFileContainsVBA Then
        VBAflag = 0
    Else
        VBAflag = 1
    End If

    Call ThisDisplay.FileSaveAsType( _
        ThisDisplay.GetFileName, _
        VBAflag)
```

Sub GetClientDimensionsPixels

```
Sub GetClientDimensionsPixels(left As Long, _
     top As Long, _
     width As Long, _
     height As Long)
```

Gets the GraphWorX32 client rectangle's size and location in pixels. The client rectangle is the area of the main window not including the borders, title bar, and menu bar.

VBA example:

```
    Dim left As Long
    Dim top As Long
    Dim width As Long
```

```
    Dim height As Long

    Call ThisDisplay.GetClientDimensionsPixels( _
                                            left, _
                                            top, _
                                            width, _
                                            height)

    MsgBox "Client dimensions in pixels: " + _
        Chr(13) + _
        "left: " + CStr(left) + Chr(13) + _
        "top: " + CStr(top) + Chr(13) + _
        "width: " + CStr(width) + Chr(13) + _
        "height: " + CStr(height)
```

VB example:

```
    Dim Left
    Dim top
    Dim width
    Dim height

    Call ThisDisplay.GetClientDimensionsPixels2( Left, _
                                                 top, _
                                                 width, _
                                                 height)

    MsgBox "Client dimensions in pixels: " + _
        Chr(13) + _
        "left: " + CStr(Left) + Chr(13) + _
        "top: " + CStr(top) + Chr(13) + _
        "width: " + CStr(width) + Chr(13) + _
        "height: " + CStr(height)
```

Sub GetDisplayDimensions, Sub GetDisplayDimensions2

```
Sub GetDisplayDimensions(width As Long, height As Long)
```

Retrieves the display dimensions (work area/world bounds) of the currently loaded display.

VBA example:

```
    'declare coordinates as long values
    Dim dispWidth As Long
    Dim dispHeight As Long

    'By reference values we get coordinates:
    Call ThisDisplay.GetDisplayDimensions(dispWidth, _
                                          dispHeight)

    MsgBox "The display dimensions are:" + _
       Chr(13) + " Width = " + _
       Conversion.CStr(dispWidth) + _
       ", Height = " + _
       Conversion.CStr(dispHeight)
```

Note: In VB scripts use subroutine GetDisplayDimensions2.

VB example:

```
    'declare coordinates as long values
    Dim dispWidth As Long
```

```
    Dim dispHeight As Long

    'TO DO!
    'By reference values we get coordinates:
    Call ThisDisplay.GetDisplayDimensions2(dispWidth, _
                                           dispHeight)

    MsgBox "The display dimensions are:" + _
       Chr(13) + " Width = " + _
       CStr(dispWidth) + _
       ", Height = " + _
       CStr(dispHeight)
```

Function GetDynamicObjectFromName

```
Function GetDynamicObjectFromName(objectName As String) As Object
```

It gets the dynamic object with the specified object name. The returned object is of type **GwxDynamic** or one of the **GwxDynamic** derived types (**GwxSize**, **GwxLocation**, **GwxRotation**, **GwxHide**, **GwxFlash**, **GwxPick**, **GwxDigitalColor**, **GwxAnalogColor**, **GwxAnimator**, **GwxAnalogSelector**, **GwxDigitalSelector**, **GwxProcessPoint**, **GwxTimedate**). Returns NULL if no matching object is found.

VBA example:

```
    Dim obj As Object
    Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

    Dim oProcessPoint As GwxProcessPoint
    Set oProcessPoint = obj
    'You can complete your code with intellisence.

    'We are going to change
    oProcessPoint.dataSource = "gfwsim.random.long"
```

Function GetEventScriptText

```
Function GetEventScriptText(Event As GWXEVENTNAME) As String
```

GraphWorX32 allows for associating VBScript and JScript with events. The scripts associated with an event are executed when the event is triggered (for example, after runtime starts, or before animating a layer). These scripts can be edited through the new integrated script toolbar or through the OLE automation interface. Each script can be written using VBScript or JScript language independently from the other event. This function allows for reading (not changing) the script source code currently associated with the specified event.

```
Enum GWXEVENTNAME:
enum {
    DataEntryActivated = 0
    DataEntryDeactivated = 1
    DataEntryValueEntered = 2
    DialBeginDrag = 3
    DialDrag = 4
    DialEndDrag = 5
    DisplayLoad = 6
    DisplayUnload = 7
    LanguageChanged = 8
    PickPostDown = 9
    PickPostUp = 10
    PickPostWhileDown = 11
```

```
    PickPreDown = 12
    PickPreUp = 13
    PickPreWhileDown = 14
    PostAnimateDisplay = 15
    PostDeanimateDisplay = 16
    PreAnimateDisplay = 17
    PreDeanimateDisplay = 18
    PreAnimateLayer = 19
    PostAnimateLayer = 20
    PreDeanimateLayer = 21
    PostDeanimateLayer = 22
    PreRuntimeStart = 23
    PostRuntimeStart = 24
    PreRuntimeStop = 25
    PostRuntimeStop = 26
    SliderBeginDrag = 27
    SliderEndDrag = 28
    SliderDrag = 29
    VCRStart = 30
    VCRStop = 31
    GlobalAliasChanged = 32
    ViewChanged = 33
    DisplayChanged = 34
    WindowChanged = 35
    PrePrint = 36
    PostPrint = 37
} GWXEVENTNAME;
```

VBA example:

```
    MsgBox ThisDisplay.GetEventScriptText(PickPostDown)
```

VB example:

```
    MsgBox ThisDisplay.GetEventScriptText(9)
```

Function GetEventScriptType

```
Function GetEventScriptType(Event As GWXEVENTNAME) As GWXSCRIPTTYPE
```

GraphWorX32 7.0 allows for associating VBScript and JScript with events. The scripts associated with an event are executed when the event is triggered (for example, after runtime starts, or before animating a layer). These scripts can be edited through the new integrated script toolbar or through the OLE automation interface. Each script can be written using VBScript or JScript language independently from the other event. This function allows for reading (not changing) the script source code currently associated with the specified event.

```
Enum GWXSCRIPTTYPE
    ScriptTypeNone = -1
    ScriptTypeVBA = 0
    ScriptTypeVBScript = 1
    ScriptTypeJScript = 2
```

VBA example:

```
'works in configure and runtime mode
'creates VB script on event PostRuntimeStart

'sets PostRuntimeStart event script type to VB script
ThisDisplay.SetEventScriptType _
PostRuntimeStart, ScriptTypeVBScript

'adds text to the script
ThisDisplay.SetEventScriptText PostRuntimeStart, _
"MsgBox " & Chr(34) & "Test OK" & Chr(34)
MsgBox ThisDisplay.GetEventScriptType(PostRuntimeStart)
MsgBox ThisDisplay.GetEventScriptText(PostRuntimeStart)
```

Function GetFileContainsVBA

```
Function GetFileContainsVBA() As Boolean
```

The function **GetFileContainsVBA()** returns True value if the display currently contains the VBA part and False if the display does not contain the VBA part. The presence of the VBA part is controlled by loading a non-VBA display and saving the display as non-VBA, or by using the **FileSaveAsType(1)** (contains VBA) or **FileSaveAsType(0)** (does not contain VBA) automation methods.

VBA example:

```
    If ThisDisplay.GetFileContainsVBA Then
        MsgBox "This display currently contains the VBA part."
    End If
```

VB example:

```
    If ThisDisplay.GetFileContainsVBA Then
        MsgBox "This display currently contains the VBA part."
    End If
```

Function GetFileName

```
Function GetFileName() As String
```

Function returns the name of the currently loaded display.

VBA example:

```
    MsgBox "The name of this file is: " + _
        ThisDisplay.GetFileName
```

VB example:

```
    MsgBox "The name of this file is: " + _
        ThisDisplay.GetFileName
```

Function GetFilePath

```
Function GetFilePath() As String
```

Returns the path where the currently loaded display is located. The returned path does not include the file name of the currently loaded display.

VBA example:

```
    Dim strPath As String
```

```
    strPath = ThisDisplay.GetFilePath

    MsgBox "The file path of the current display is: " + _
        strPath
```

VB example:

```
    Dim strPath

    strPath = ThisDisplay.GetFilePath

    MsgBox "The file path of the current display is: " + _
        strPath
```

Function GetFileVersion

```
Function GetFileVersion() As Long
```

The function **GetFileVersion()** returns the following values:

-1 .. OpenFileError
0 .. Ver520
1 .. Ver600
2 .. Ver600SP
3 .. VerMGraphics40
4 .. Ver610
5 .. Ver620
9999 .. VerLatest

VBA example:

```
    'Get the version of the current
    'release.
    Dim lFileVersion As Long
    lFileVersion = ThisDisplay.GetFileVersion

    Dim strVersion As String
    Select Case lFileVersion
        Case -1
            strVersion = "OpenFileError"
        Case 0
            strVersion = "Ver520"
        Case 1
            strVersion = "Ver600"
        Case 2
            strVersion = "Ver600SP"
        Case 3
            strVersion = "VerMGraphics40"
        Case 4
            strVersion = "Ver610"
        Case 8 To 9999 'year 2004
            strVersion = "VerLatest"
    End Select

    MsgBox "The file version is: " + _
        CStr(lFileVersion) + _
        "(" + strVersion + ")"
```

VB example:

```
    'Get the version of the current
```

```
    'release.
    Dim lFileVersion
    lFileVersion = ThisDisplay.GetFileVersion

    Dim strVersion
    Select Case lFileVersion
        Case -1
            strVersion = "OpenFileError"
        Case 0
            strVersion = "Ver520"
        Case 1
            strVersion = "Ver600"
        Case 2
            strVersion = "Ver600SP"
        Case 3
            strVersion = "VerMGraphics40"
        Case 4
            strVersion = "Ver610"
        Case 8
            strVersion = "VerLatest"
    End Select

    MsgBox "The file version is: " + _
        CStr(lFileVersion) + _
        "(" + strVersion + ")"
```

Function GetFrameWindowHandle

```
Function GetFrameWindowHandle() As Long
```

Gets the window handle of the frame window in which this **GwxDisplay** object is running.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    Dim handle As Long
    handle = ThisDisplay.GetFrameWindowHandle

    'Use the handle to specify the parent window.
    Dim tagName As String
    Call ThisDisplay.OpenTagBrowser(handle, tagName)
```

VB example:

```
    Dim handle
    handle = ThisDisplay.GetFrameWindowHandle

    'Use the handle to specify the parent window.
    Dim tagName
    Call ThisDisplay.OpenTagBrowser(handle, tagName)
```

Function GetGASThemes

```
Function GetGASThemes() As String
```

Gets the currently set Global Aliasing themes in the same format as generated by the Theme browser.

VBA example:

```
'VBA/VB Script example, works only in runtime mode
'example for setting and getting themes (Floors theme is changed in _
child windows)
```

```
ThisDisplay.SetGASThemes _
"Buildings=BuildingB;Floors=Floor2/+1;"
MsgBox ThisDisplay.GetGASThemes
```

Function GetHeadObject

```
Function GetHeadObject() As Object
```

Gets the head visible object. In configure mode, the head object is the selected object with the solid blue or red resize grips. In runtime mode, the head object is the pickable object which currently has focus. If there is no head object, this method returns NULL.

VBA example:

```
    Dim obj As Object
    Set obj = ThisDisplay.GetHeadObject

    MsgBox "The head visible object is: " + _
        obj.objectName
```

VBA example in configure mode:

```
   Dim obj As Object
    Set obj = ThisDisplay.GetHeadObject

    MsgBox "The head visible object is: " + _
        obj.objectName
```

Function GetLoadTab, GetLoadTab2

```
Function GetLoadTab(index As Long, _
     label As String, _
     File As String, _
     AliasCmdType As Long, _
     AliasCmdString As String, _
     GlobalAliasCmdString As String) As Boolean
```

Returns TRUE when LoadTab information is successfully retrieved. The index parameter is zero-based index of the load tab. Load Tab information is returned via the set of remaining parameters involved.

VBA example:

```
    Dim strlabel As String
    Dim strfile As String
    Dim lCommandType As Long
    Dim strAliasCommand As String
    Dim strGlAliasCommand As String

    If ThisDisplay.GetLoadTab(0, _
                              strlabel, _
                              strfile, _
                              lCommandType, _
                              strAliasCommand, _
                              strGlAliasCommand) _
                           Then
        MsgBox "Properties of the display(0): " + _
            Chr(13) + _
            "label: " + strlabel + Chr(13) + _
            "file: " + strfile + Chr(13) + _
            "AliasCmdType: " + CStr(lCommandType) + Chr(13) + _
```

```
            "AliasCmdString: " + strAliasCommand + Chr(13) + _
            "GlobalAliasCmdString: " + strGlAliasCommand
    End If
```

Function GetLoadTabsCount

```
Function GetLoadTabsCount() As Long
```

Returns number of Load Tabs in the display.

Function GetLocalAliases

```
Function GetLocalAliases(DisplayName As String)
```

Gets local aliases from the display *DisplayName*.

VBA example:

```
    Dim strAliases As Variant

    Dim gwxNewDisp As New Gwx32.GwxDisplay

    strAliases = gwxNewDisp.GetLocalAliases( _
                    ThisDisplay.GetFilePath + _
                    "\GwxDynamicVBA.gdf")

    MsgBox "Have a look in the GwxDynamicVBA.gdf file..."
    MsgBox "The first alias is: " + _
            strAliases(0)
```

VB example:

```
    Dim strAliases
    Dim newGwxDisplay
    Set newGwxDisplay = CreateObject("Gwx32.Display")


    strAliases = newGwxDisplay.GetLocalAliases( _
                    ThisDisplay.GetFilePath + _
                    "\GwxDynamicVBA.gdf")

    MsgBox "Have a look in the GwxDynamicVBA.gdf file..."
    MsgBox "The first alias is: " + _
            strAliases(0)
```

Function GetLocalVariables

```
Function GetLocalVariables(DisplayName As String)
```

Gets local variables from the display *DisplayName*.

VBA example:

```
    Dim strVars As Variant

    Dim gwxNewDisp As New Gwx32.GwxDisplay
    strVars = gwxNewDisp.GetLocalVariables( _
                    ThisDisplay.GetFilePath + _
                    "\GwxDynamicVBA.gdf")

    MsgBox "Have a look in the GwxDynamicVBA.gdf file..."
    MsgBox "The first variable is: " + _
            strVars(0)
```

VB example:

```
Dim strVars

Dim newGwxDisplay
Set newGwxDisplay = CreateObject("Gwx32.Display")

strVars = newGwxDisplay.GetLocalVariables( _
                ThisDisplay.GetFilePath + _
                "\GwxDynamicVBA.gdf")

MsgBox "Have a look in the GwxDynamicVBA.gdf file..."
MsgBox "The first local alias is: " + _
        strVars(0)
```

Function GetNumberOfTopLevelVisibles

```
Function GetNumberOfTopLevelVisibles() As Long
```

Returns the number of visible objects in the root symbol of a display (referred to as “Top Level Visibles” because they are the objects in top-most level of the symbol hierarchy).

VBA example:

```
Dim visObj As Long
'returns the number of visible objects:
visObj = ThisDisplay.GetNumberOfTopLevelVisibles

MsgBox "The number of visible objects is: " + _
    CStr(visObj)
```

VB example:

```
Dim visObj
'returns the number of visible objects:
visObj = ThisDisplay.GetNumberOfTopLevelVisibles

MsgBox "The number of visible objects is: " + _
    CStr(visObj)
```

Function GetObjectTypeName

```
Function GetObjectTypeName() As String
```

Returns the type name of this object (“**GwxDisplay**”).

VBA example:

```
'We assume you have created a rectangle
'with the name "Rectangle".
Dim strObjectName As String

'Get the name of this display:
strObjectName = ThisDisplay.GetObjectTypeName
MsgBox "ThisDisplay.GetObjectTypeName returns: " + _
    strObjectName

'Get the type name of the object "Rectangle":
Dim obj As Object
Set obj = ThisDisplay.GetVisibleObjectFromName("Rectangle")
strObjectName = obj.GetObjectTypeName
MsgBox "The object type name of the rectangle above is: " + _
    strObjectName
```

VB example:

```
'We assume you have created a rectangle
'with the name "Rectangle".
Dim strObjectName

'Get the name of this display:
strObjectName = ThisDisplay.GetObjectTypeName
MsgBox "ThisDisplay.GetObjectTypeName returns: " + _
    strObjectName

'Get the type name of the object "Rectangle":
Dim obj
Set obj = ThisDisplay.GetVisibleObjectFromName("Rectangle")

strObjectName = obj.GetObjectTypeName
MsgBox "The object type name of the rectangle above is: " + _
    strObjectName
```

Function GetOpenPopupByIndex

```
Function GetOpenPopupByIndex(popupIndex As Long) As Object
```

Returns the **GwxDisplay** object of the open popup or embedded popup window with the specified zero-based index (**popupIndex**, indexes the list of popups and embedded popups owned by this display).

Note: Not available in ActiveX ToolWorX.

VBA example:

```
'Index number:
'#0
Call ThisDisplay.OpenPopupWindow("PopupWindow.gdf", _
                                 False, _
                                 True, _
                                 False)
'#1
Call ThisDisplay.OpenPopupWindow("PopupWindow.gdf", _
                                 False, _
                                 False, _
                                 False)
Dim obj As Object
Set obj = ThisDisplay.GetOpenPopupByIndex(1)
Dim oPopup As GwxDisplay
Set oPopup = obj
oPopup.BackgroundColor = RGB(255, 0, 0)
```

VB example:

```
'Index number:
'#0
Call ThisDisplay.OpenPopupWindow("PopupWindow.gdf", _
                                 False, _
                                 True, _
                                 False )
'#1
Call ThisDisplay.OpenPopupWindow("PopupWindow.gdf", _
                                 False, _
                                 False, _
                                 False)
Dim obj
Set obj = ThisDisplay.GetOpenPopupByIndex(1)
```

```
obj.BackgroundColor = RGB(255, 0, 0)
```

Function GetOpenPopupCount

```
Function GetOpenPopupCount() As Long
```

Returns the number of open popup and embedded popup windows owned by this display.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
Call ThisDisplay.OpenPopupWindow("PopupWindow.gdf", _
                                 False, _
                                 True, _
                                 False)
MsgBox "The number of popupWindows is " + _
        CStr(ThisDisplay.GetOpenPopupCount) + "." + _
        Chr(13) + "Click on the button once more please."
```

VB example:

```
Call ThisDisplay.OpenPopupWindow("PopupWindow.gdf", _
                                 False, _
                                 True, _
                                 False)
MsgBox "The number of popupWindows is " + _
        CStr(ThisDisplay.GetOpenPopupCount) + "." + _
        Chr(13) + "Click on the button once more please."
```

Function GetOpenPopupWithTitle

```
Function GetOpenPopupWithTitle(titleSubstring As String) As Object
```

Gets the open popup window's display object which has the specified substring as part or all of the text in the popup window's title bar. Returns the object of the display loaded into the popup window (of type **GwxDisplay**) on success, *Nothing* on failure.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
Dim parentDisplay As Object

'Assign an object to popup
Set popup = ThisDisplay.GetOpenPopupWithTitle("PopupWindow.gdf")

If Nothing Is popup Then
    MsgBox "There is no popup."
Else
    MsgBox "The path to the popup window is:" + _
    popup.GetFileName
End If
```

VB example:

```
Dim popup

'Assign an object to popup
 Set popup = ThisDisplay.GetOpenPopupWithTitle("PopupWindow.gdf")

 If Nothing Is popup Then
```

```
        MsgBox "There is no popup."
    Else
        MsgBox "The path to the popup window is:" + _
        popup.GetFileName
    End If
```

Function GetParentDisplay

```
Function GetParentDisplay() As Object
```

If this display object is running in a popup window or embedded window, this method returns the display object of the parent display (the parent display is the display that opened the popup window). This method returns *Nothing* if the display is not running in a popup/embedded window.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    Dim parentDisplay As Object

    'Assign an object to parentDisplay
    Set parentDisplay = ThisDisplay.GetParentDisplay

    If Nothing Is parentDisplay Then
        MsgBox "There is no parent window."
    Else
        MsgBox "The path to the parent display is:" + _
        parentDisplay.GetFileName
    End If
```

VB example:

```
    Dim parentDisplay

    'Assign an object to parentDisplay
    Set parentDisplay = ThisDisplay.GetParentDisplay

    If Nothing Is parentDisplay Then
        MsgBox "There is no parent window."
    Else
        MsgBox "The path to the parent display is:" + _
        parentDisplay.GetFileName
    End If
```

Function GetPointObjectFromName

```
Function GetPointObjectFromName(pointName As String) As Object
```

Gets the data point object with the specified point name. The returned object is of type **GwxPoint**. Returns NULL if no matching object is found.

VBA example:

```
   Dim obj As Object
    Set obj = _
     ThisDisplay.GetPointObjectFromName("gfwsim.cosine.double")

    Dim oPoint As GwxPoint
    Set oPoint = obj

    Dim strName As String
    Dim strType As String
```

```
    Dim strPointName As String
    Dim val As Double
    Dim low As Double
    Dim high As Double

    strName = oPoint.GetObjectTypeName
    strType = oPoint.dataType
    strPointName = oPoint.GetPointName
    high = oPoint.highRange
    low = oPoint.lowRange
    val = oPoint.Value

    MsgBox "Source gfwsim.cosine.double: " + _
        "Object type name: " + strName + Chr(13) + _
        "Data Type: " + strType + Chr(13) + _
        "Point Name: " + strPointName + Chr(13) + _
        "High Range: " + CStr(high) + Chr(13) + _
        "Low Range: " + CStr(low) + Chr(13) + _
        "Current Value: " + CStr(val)
```

VB example:

```
    Dim oPoint
    Set oPoint = _
       ThisDisplay.GetPointObjectFromName("gfwsim.cosine.double")

    Dim strName
    Dim strType
    Dim strPointName
    Dim val
    Dim low
    Dim high

    If oPoint Is Nothing Then
       MsgBox "There is no source gfwsim.cosine.double to render."
    End If

    strName = oPoint.GetObjectTypeName
    strType = oPoint.dataType
    strPointName = oPoint.GetPointName
    high = oPoint.highRange
    low = oPoint.lowRange
    val = oPoint.Value

    MsgBox "Source gfwsim.cosine.double: " + _
        "Object type name: " + strName + Chr(13) + _
        "Data Type: " + strType + Chr(13) + _
        "Point Name: " + strPointName + Chr(13) + _
        "High Range: " + CStr(high) + Chr(13) + _
        "Low Range: " + CStr(low) + Chr(13) + _
        "Current Value: " + CStr(val)
```

Function GetPrintViewArea

```
Function GetPrintViewArea()
```

Gets the view area of the display ready to print.

Enumerate:
PV_NOCHANGE = 0,
PV_HOME = 1,
PV_FITTOWINDOW = 2,

PV_WORLD = 3

VBA example:

```
Dim printViewArea As Boolean
printViewArea = ThisDisplay.GetPrintViewArea

Dim strprintViewArea As String
Select Case printViewArea
    Case 0
        strprintViewArea = "PV_NOCHANGE"
    Case 1
        strprintViewArea = "PV_HOME"
    Case 2
        strprintViewArea = "PV_FITTOWINDOW"
    Case 3
        strprintViewArea = "PV_WORLD"
End Select

MsgBox "The layout for printing is: " + _
    StrprintViewArea
```

VB example:

```
Dim printViewArea
printViewArea = ThisDisplay.GetPrintViewArea

Dim strprintViewArea
Select Case printViewArea
    Case 0
        strprintViewArea = "PV_NOCHANGE"
    Case 1
        strprintViewArea = "PV_HOME"
    Case 2
        strprintViewArea = "PV_FITTOWINDOW"
    Case 3
        strprintViewArea = "PV_WORLD"
End Select

MsgBox "The layout for printing is: " + _
    strprintViewArea
```

Function GetPrintWhiteBackground

```
Function GetPrintWhiteBackground () As Long
```

Gets the print view of the current display.

VBA example:

```
Dim layout As Boolean
layout = ThisDisplay.GetPrintWhiteBackground


MsgBox "White background: " + CStr(layout)
```

VB example:

```
 Dim layout
layout = ThisDisplay.GetPrintWhiteBackground
```

```
    MsgBox "White background: " + CStr(layout)
```

Function GetSimulationVariables

```
Function GetSimulationVariables(DisplayName As String)
```

Obtains the variables from another display.

VBA example:

```
    Dim strVars As Variant

    Dim gwxNewDisp As New Gwx32.GwxDisplay
    MsgBox "Before you use this function you have " + _
            Chr(13) + "to specify the path to display " + _
            "that you are interested in. e.g.” + _
            “\\SERVER\...\myDisplay.gdf"

    'strVars = _
gwxNewDisp.GetSimulationVariables("\\SERVER\...\GwxDynamicVBA.gdf")

    'MsgBox "The first variable is: " + _
    '        strVars(0)
```

VB example:

```
    Dim strVars

    Dim newGwxDisplay

    Set newGwxDisplay = CreateObject("Gwx32.Display")

    MsgBox "Before you use this function you have " + _
            Chr(13) + "to specify the path to display " + _
            "that you are interested in. e.g.” + _
           “\\SERVER\...\myDisplay.gdf"

    'strVars = _
newGwxDisplay.GetSimulationVariables("\\SERVER\...\GwxDynamicVBA.gdf")

    'MsgBox "The first variable is: " + _
    '        strVars(0)
```

Function GetSystemWideLanguage

```
Function GetSystemWideLanguage(Language As String, _
     LanguageSubset As String, _
     LocaleID As Long, _
     SynchronizeLCID As Boolean) As Boolean
```

Gets the currently selected language for GENESIS applications

VBA example:

```
    Dim strLanguage As String
    Dim strLanguageSubSet As String
    Dim lLocaleID As Long
    Dim bSynchronizeLCID As Boolean

    Call ThisDisplay.GetSystemWideLanguage(strLanguage, _
```

```
                                                strLanguageSubSet, _
                                                lLocaleID, _
                                                bSynchronizeLCID)
```

Function GetTransparency, GetTransparency2

```
Function GetTransparency(TransparencyEnabled As Boolean, _
     transparentColor As OLE_COLOR, _
     AlphaEnabled As Boolean, _
     AlphaBlending As Integer) As Boolean
```

Gets information about the transparency settings. This function works only in GraphWorX32 on Windows 2000 and higher.

VBA example:

```
'VBA example, works in configure and runtime mode
'this VBA example gets transparency and transluency settings

Dim  TE As Boolean, _
TC As OLE_COLOR, _
AE As Boolean, _
AB As Integer
ThisDisplay.GetTransparency TE, TC, AE, AB
MsgBox "Transparency: " & TE & vbCrLf & "Color: " & _
TC & vbCrLf & "Transluency: " & AE & _
vbCrLf & "Color: " & AB
```

VB example:

```
Dim TE
Dim TC
Dim AE
Dim AB

Call ThisDisplay.GetTransparency2( TE, TC, AE, AB)

MsgBox "Transparency: " & TE & vbCrLf & "Color: " & _
TC & vbCrLf & "Transluency: " & AE & _
vbCrLf & "Color: " & AB
```

Function GetVCRMode

```
Function GetVCRMode() As Boolean
```

Gets the state of VCR mode. It indicates whether the data can be replayed from the HDA database or not.

VBA example:

```
    'See the examples in
    '...\GENESIS-32\Examples\VCRWorX32 Examples\
    'directory.

    If ThisDisplay.GetVCRMode Then
        MsgBox "VCR mode is on."
    Else
        MsgBox "VCR mode is off."
    End If
```

VB example:

```
'See the examples in
'...\GENESIS-32\Examples\VCRWorX32 Examples\
'directory.

If ThisDisplay.GetVCRMode Then
    MsgBox "VCR mode is on."
Else
    MsgBox "VCR mode is off."
End If
```

void GetViewDimensions, void GetViewDimensions2

```
void GetViewDimensions(long* left, _
     long* top, _
     long* width, _
     long* height)
```

It gets the GraphWorX32 view rectangle's size and location.

VBA example:

```
'declare coordinates as long values
Dim viewLeft As Long
Dim viewTop As Long
Dim viewWidth As Long
Dim viewHeight As Long

'By reference values we get coordinates:
Call ThisDisplay.GetViewDimensions(viewLeft, _
                                   viewTop, _
                                   viewWidth, _
                                   viewHeight)

MsgBox "The view dimensions are:" + _
  Chr(13) + " Left = " + _
  Conversion.CStr(viewLeft) + _
  ", Top = " + _
  Conversion.CStr(viewTop) + _
  ", Width = " + _
  Conversion.CStr(viewWidth) + _
  ", Height = " + _
  Conversion.CStr(viewHeight)
```

Note: In VB scripts use subroutine GetViewDimensions2.

VB example:

```
'declare coordinates as long values
 Dim viewLeft
 Dim viewTop
 Dim viewWidth
 Dim viewHeight

 'TO DO!
 'By reference values we get coordinates:
 Call ThisDisplay.GetViewDimensions2(viewLeft, _
                                     viewTop, _
                                     viewWidth, _
```

```
                                        viewHeight)

    MsgBox "The view dimensions are:" + _
      Chr(13) + " Left = " + _
      CStr(viewLeft) + _
      ", Top = " + _
      CStr(viewTop) + _
      ", Width = " + _
      CStr(viewWidth) + _
      ", Height = " + _
      CStr(viewHeight)
```

Function GetVisibleObjectFromIndex

```
Function GetVisibleObjectFromIndex(index As Long) As Object
```

Returns the Top Level Visible with the given zero-based index. An object with the an index of zero is the object furthest back in the z-order. This function is useful for iterating through all the top level visible objects.

VBA example:

```
    'Indexes.
    Dim counter
    counter = 0

    'A list with object names.
    Dim list As String

    'An object.
    Dim obj As Object
    Set obj = ThisDisplay.GetVisibleObjectFromIndex(counter)
    list = obj.objectName

    'Keep going ...
    Do
        counter = counter + 1
        If obj.objectName <> "" Then
            list = list + Chr(13) + obj.objectName
        End If
        Set obj = ThisDisplay.GetVisibleObjectFromIndex(counter)
    Loop Until obj Is Nothing

    MsgBox "Complete list of named objects in the display: " + _
        Chr(13) + list
```

VB example:

```
    'Indexes.
    Dim counter
    counter = 0

    'A list with object names.
    Dim list

    'An object.
    Dim obj
    Set obj = ThisDisplay.GetVisibleObjectFromIndex(counter)
    list = obj.objectName
```

```
    'Keep going ...
    Do
        counter = counter + 1
        If obj.objectName <> "" Then
            list = list + Chr(13) + obj.objectName
        End If
        Set obj = ThisDisplay.GetVisibleObjectFromIndex(counter)
    Loop Until obj Is Nothing

   MsgBox "Complete list of named objects in the display: " + _
        Chr(13) + list
```

Function GetVisibleObjectFromName

```
Function GetVisibleObjectFromName(objectName As String) As Object
```

Gets the visible object with the specified object name. The returned object type will be **GwxVisible** or one of **GwxVisible** derived object types (**GwxRectangle**, **GwxEllipse**, **GwxLine**, **GwxText**, **GwxArc**, **GwxBitmap**, **GwxMetafile**, **GwxSymbol**, **GwxOleObject**, **GwxButton**). Returns NULL if no matching object is found.

VBA example:

```
    'If you do not know the actual type
    'of an object and you want to access
    'it generically, use the object type
    'Object.  This will cause VB to access
    'methods and properties of the object
    'by name rather than by ID, thereby
    'avoiding ID numbering issues.

    Dim obj As Object
    'Dim rect As GwxRectangle 'is optional
    'Set rect = obj

    Set obj = ThisDisplay.GetVisibleObjectFromName("Rectangle")

    MsgBox "Change the color of the rectangle above." + _
        " Its name is: " + obj.objectName

    obj.fillColor = rgb(200, 200, 255)
    MsgBox "Undo fillcolor."
    obj.fillColor = rgb(255, 191, 0)
```

VB example:

```
    'If you do not know the actual type
    'of an object and you want to access
    'it generically, use the object type
    'Object.  This will cause VB to access
    'methods and properties of the object
    'by name rather than by ID, thereby
    'avoiding ID numbering issues.

    Dim obj
    'Dim rect As GwxRectangle 'is optional
    'Set rect = obj

    Set obj = ThisDisplay.GetVisibleObjectFromName("Rectangle")

    MsgBox "Change the color of the rectangle above." + _
        " Its name is: " + obj.objectName
```

```
    obj.fillColor = RGB(200, 200, 255)
    MsgBox "Undo fillcolor."
    obj.fillColor = RGB(255, 191, 0)
```

Function GetVisibleObjectFromPosition

```
Function GetVisibleObjectFromPosition(x As Long, _
     y As Long,_
     GoIntoSymbol As Boolean) As Object
```

Returns an object at a specified position in the display. Starts from the root of the display and goes deeper, including layers, until the top-most object is found. If the argument **GoIntoSymbol** is set to True, the searching continues until the final visible object at a given position. Otherwise the function returns the top-level symbol that contains the object.

VBA example:

```
'Works in configure and runtime mode
'this example gets the lowest level visible at random position

Dim w As Long, h As Long
Dim o As Object
ThisDisplay.GetDisplayDimensions w, h
'gets display width and height
If ThisDisplay.GetNumberOfTopLevelVisibles > 0 Then
'checks if display has at least one visible object!
'and runs a loop for find one

Do
  Set o = ThisDisplay.GetVisibleObjectFromPosition( _
Rnd * w, Rnd * h, True)

'tries to get visible object from random position - _
!if object is GwxSymbol, it goes into symbol and _
!continues searching

Loop Until Not (o Is Nothing)

'after any object is found, loop ends and example _
!shows name of the object
 MsgBox o.objectName
Else
 MsgBox "Example needs at least one visible object"
End If
```

VBA example:

```
    Dim obj As Object
    'We assume you have created a rectangle on the
    'specific position.
    Set obj = ThisDisplay.GetVisibleObjectFromPosition(588, _
                                                       256, _
                                                       True)
    Dim rect As GwxRectangle
    Set rect = obj
    MsgBox "Change the color of the rectangle." + _
        " Its name is: " + CStr(rect.objectName)

    rect.fillColor = rgb(200, 200, 255)
    MsgBox "Undo fillcolor."
    rect.fillColor = rgb(255, 191, 0)
```

VB example:

```
    Dim obj

    Set obj = ThisDisplay.GetVisibleObjectFromPosition(588, _
                                                       256, _
                                                       True)
    Dim rect
    Set rect = obj
    MsgBox "Change the color of the rectangle." + _
        " Its name is: " + CStr(rect.objectName)

    rect.fillColor = RGB(200, 30, 255)

    MsgBox "Undo fill color."

    rect.fillColor = RGB(200, 200, 255)
```

Sub GetWindowDimensionsPercent, Sub GetWindowDimensionsPercent2

```
Sub GetWindowDimensionsPercent(left As Single, _
     top As Single, _
     width As Single, _
     height As Single)
```

Gets the GraphWorX32 main window size and location as a percentage of the total screen size. Retrieved parameters will have values in the range of 0.0 to 1.0.

Note: In VB scripts use subroutine GetWindowDimensionsPercent2.

VBA example:

```
Sub GetWndDimPercent(o As GwxPick)
    'declare resolution as long values
    Dim percLeft As Single
    Dim percTop As Single
    Dim percWidth As Single
    Dim percHeight As Single

    'By reference values we can get dimensions:
    Call ThisDisplay.GetWindowDimensionsPercent(percLeft, _
                                                percTop, _
                                                percWidth, _
                                                percHeight)

    MsgBox "GraphWorX32 main window size in percent is:" + _
    Chr(13) + " Left = " + _
    Conversion.CStr(percLeft) + _
    ", Top = " + _
    Conversion.CStr(percTop) + _
    ", Width = " + _
    Conversion.CStr(percWidth) + _
    ", Height = " + _
    Conversion.CStr(percHeight)

End Sub
```

VB example:

```
    'declare resolution
    Dim percLeft
    Dim percTop
```

```
    Dim percWidth
    Dim percHeight

    'By reference values we can get dimensions:
    Call ThisDisplay.GetWindowDimensionsPercent2(percLeft, _
                                                 percTop, _
                                                 percWidth, _
                                                 percHeight)

    MsgBox "GraphWorX32 main window size in percent is:" + _
    Chr(13) + " Left = " + _
    CStr(percLeft) + _
    ", Top = " + _
    CStr(percTop) + _
    ", Width = " + _
    CStr(percWidth) + _
    ", Height = " + _
    CStr(percHeight)
```

Sub GetWindowDimensionsPixels, Sub GetWindowDimensionsPixels2

```
Sub GetWindowDimensionsPixels(left As Long, _
     top As Long, _
     width As Long, _
     height As Long)
```

Gets the GraphWorX32 main window size and location in pixels.

VBA example:

```
Sub GetWndDimPixels(o As GwxPick)
    'declare resolution as long values
    Dim pixLeft As Long
    Dim pixTop As Long
    Dim pixWidth As Long
    Dim pixHeight As Long


    'By reference values we can get dimensions:
    Call ThisDisplay.GetWindowDimensionsPixels(pixLeft, _
                                               pixTop, _
                                               pixWidth, _
                                               pixHeight)

    MsgBox "GraphWorX32 main window size in pixels is:" + _
    Chr(13) + " Left = " + _
    Conversion.CStr(pixLeft) + _
    ", Top = " + _
    Conversion.CStr(pixTop) + _
    ", Width = " + _
    Conversion.CStr(pixWidth) + _
    ", Height = " + _
    Conversion.CStr(pixHeight)

End Sub
```

VBA example:

```
   'Declare resolution as long values
    Dim pixLeft
    Dim pixTop
```

```
    Dim pixWidth
    Dim pixHeight

    TO DO!
    'By reference values we can get dimensions:
    Call ThisDisplay.GetWindowDimensionsPixels2(pixLeft, _
                                                pixTop, _
                                                pixWidth, _
                                                pixHeight)

    MsgBox "GraphWorX32 main window size in pixels is:" + _
    Chr(13) + " Left = " + _
    CStr(pixLeft) + _
    ", Top = " + _
    CStr(pixTop) + _
    ", Width = " + _
    CStr(pixWidth) + _
    ", Height = " + _
    CStr(pixHeight)
```

Function GetWorkingDirectory

```
Function GetWorkingDirectory() As String
```

Returns the GENESIS32 working directory. This is the directory where GENESIS32 applications like GraphWorX32 look for display files and other configuration files.

VBA example:

```
    MsgBox "The working directory is: " + _
        ThisDisplay.GetWorkingDirectory
```

VB example:

```
    MsgBox "The working directory is: " + _
        ThisDisplay.GetWorkingDirectory
```

Function GwxGetVbaProject

```
Function GwxGetVbaProject() As Object
```

Returns the VBA Project interface. .

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
    'You need to add a Project | References | button Browse and
    'locate .../Program Files/Common/Microsoft...
    '...Shared/VBA/VBA6/VBE6EXT.OLB file.
    'Add it into references and check it.
    'It should point to Microsoft for Visual Basic Extensibility.

    Dim x As VBIDE.VBProject
    Set x = ThisDisplay.GwxGetVBAProject()

    ' Add a string programmatically above this subroutine.
    Call x.VBComponents.Item(1).CodeModule.AddFromString( _
       "' I am a comment!")
    Debug.Print x.HelpContextID
```

Sub HideWindow

```
Sub HideWindow()
```

Hides the GraphWorX32 main window.

VBA example:

```
MsgBox "Now the window will disapear on submit."

'Hide the main display
Call ThisDisplay.HideWindow

MsgBox "Now the window will occur on submit."

'Call the display back
Call ThisDisplay.ShowWindow
```

VB example:

```
MsgBox "Now the window will disapear on submit."

'Hide the main display
Call ThisDisplay.HideWindow

MsgBox "Now the window will occur on submit."

'Call the display back
Call ThisDisplay.ShowWindow
```

Function InsertLibraryObject

```
Function InsertLibraryObject(libraryObject As Object) As Boolean
```

Inserts a Symbol Library object into display. This library object can be created/obtained via Symbol Library OLE automation.

Note: See the Symbol Library Help documentation for more information.
Note: This function works in GraphWorX32 design mode only.

VBA example:

```
'We assume that the SymbolLibrary object
'is already loaded into your project.

Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary

MsgBox sl.GetRootDirectory

If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Misc")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

Dim symbol As Object
Set symbol = sc.GetSymbolFromName("Check")
If symbol Is Nothing Then
  MsgBox "Failed to get symbol from name!"
```

```
    Exit Sub

  End If

  If Not ThisDisplay.InsertLibraryObject(symbol) Then
    MsgBox "Error with InsertLibraryObject!"
  Else
    MsgBox "Library object successfully inserted!"
  End If
```

Function InsertLibraryObjectByIndex

```
Function InsertLibraryObjectByIndex(category As String, _
    index As Long) As Boolean
```

Allows easy inserting of any symbol from the Symbol Library via OLE automation. The **category** parameter corresponds with the name of the Symbol Library category file (.sdf file); the parameter **index** is zero-based index of the symbol in the category.

Note: This function works in GraphWorX32 design mode only.

VBA example:

```
  'We assume that the SymbolLibrary object
  'is already loaded into your project.

  Dim sl As SymbolLibrary.SymbolLibrary
  Dim sc As SymbolLibrary.SymbolCategory

  Set sl = New SymbolLibrary.SymbolLibrary
  Set sc = sl.GetCategoryFromIndex(1)

  MsgBox sl.GetRootDirectory

  If sc.GetNumberOfSymbols > 0 Then
      If Not ThisDisplay.InsertLibraryObjectByIndex( _
                              Dir(sc.GetCategoryName), _
                              0) Then
          MsgBox "Error with InsertLibraryObjectByIndex!"
      Else
          MsgBox "Library object successfully inserted!"
      End If
  Else
      MsgBox "Example needs at least one symbol"
  End If
```

Function InsertLibraryObjectByName

```
Function InsertLibraryObjectByName(category As String, symbol As String) As
Boolean
```

Allows easy inserting of any symbol from the Symbol Library via OLE automation. The **category** parameter corresponds to the name of the Symbol Library category file (.sdf file); the parameter **symbol** is the name of the symbol from category file.

Note: This function works in GraphWorX32 design mode only.
Typical use of this method:

VBA example:

```
Dim SL As SymbolLibrary.SymbolLibrary
```

```
Dim SC As SymbolLibrary.SymbolCategory
Dim o As Object

Set SL = New SymbolLibrary.SymbolLibrary
Set SC = SL.GetCategoryFromIndex(0)

If SC.GetNumberOfSymbols > 0 Then
     ThisDisplay.InsertLibraryObjectByIndex _
     Dir(SC.GetCategoryName), _
     Round(Rnd * SC.GetNumberOfSymbols)
     ThisDisplay.InsertLibraryObjectByName _
     Dir(SC.GetCategoryName), _
     SC.GetSymbolName(Round(Rnd * SC.GetNumberOfSymbols))
     Set o = SC.GetSymbolFromIndex( _
     Round(Rnd * SC.GetNumberOfSymbols))
     ThisDisplay.InsertLibraryObject o
Else
     MsgBox "Example needs at least one symbol"
End If
```

VBA example:

```
    'We assume that the SymbolLibrary object
    'is already loaded into your project.
    'In Microsoft Visual Basic -> Tools -> References check
    'the option SymbolLibrary.
    If Not ThisDisplay.InsertLibraryObjectByName("Components", "Batteries")
Then
        MsgBox "Error with InsertLibraryObjectByName!"
    Else
        MsgBox "Library object successfully inserted!"
    End If
```

Function InsertOleObject

```
Function InsertOleObject(progID As String, objName As String) As Boolean
```

This function allows you to create an OLE Automation object and then insert that object into your GraphWorX32 displays. This automation interface can be called through VBA or VBScript/JScript or C++. This method requires that you provide the name of the object that you wish to create. You can create any kind of object that you wish but you have to know the name. For example, ICONICS uses the following objects:

"ICONICS.SCRAlarmCtrl.1"
"ICONICS.Event32.1"
"ICONICS.Periodic32.1"
"GWXGAUGE.GWXGaugeCtrl.1"
"GWXSWITCH.GWXSwitchCtrl.1"
"TWXViewer.TWXViewerCtrl.1"
"AWXview32.AWXview32Ctrl.1"
"GWXVIEW32.GWXview32Ctrl.1"
"GWXNUMERIC.GWXNumericCtrl.1"
"GWXVESSEL.GwxVesselCtrl.1"
"ICONICS.FunctionalKeyCtrl.1"
"MarqueeLaunch.MarqueeCtl.1"
"PopupLaunch.PopupCtl.1"
“Gwx32.Display”

VBA example:

```
'ICONICS uses the following objects:
'ICONICS.SCRAlarmCtrl.1
'ICONICS.Event32.1
'ICONICS.Periodic32.1
'GWXGAUGE.GWXGaugeCtrl.1
'GWXSWITCH.GWXSwitchCtrl.1
'TWXViewer.TWXViewerCtrl.1
'AWXview32.AWXview32Ctrl.1
'GWXVIEW32.GWXview32Ctrl.1
'GWXNUMERIC.GWXNumericCtrl.1
'GWXVESSEL.GwxVesselCtrl.1
'ICONICS.FunctionalKeyCtrl.1
'MarqueeLaunch.MarqueeCtl.1
'PopupLaunch.PopupCtl.1
'Gwx32.Display

'To create object "Gwx32.Display" (e.g. in VB Script) you have
'to write this code:
'Dim newGwxDisplay
'Set newGwxDisplay = CreateObject("Gwx32.Display")


Call ThisDisplay.InsertOleObject("AWXview32.AWXview32Ctrl.1", _
     "view")
```

Function IsEmbeddedPopupWindow

```
Function IsEmbeddedPopupWindow() As Boolean
```

TRUE if this display is running in an embedded popup window; FALSE otherwise.

Note: Not available in ActiveX ToolWorX.

VBA example

```
Dim flag As Boolean

flag = ThisDisplay.IsEmbeddedPopupWindow

If flag Then
    MsgBox "The display is running in" + _
        " embedded popup window."
Else
    MsgBox "The display is not running in" + _
        " embedded popup window."
End If
```

VB example:

```
Dim flag

flag = ThisDisplay.IsEmbeddedPopupWindow

If flag Then
    MsgBox "The display is running in" + _
        " embedded popup window."
Else
    MsgBox "The display is not running in" + _
        " embedded popup window."
End If
```

Function IsModified

```
Function IsModified() As Boolean
```

Returns TRUE is the display has been modified since the last time it was saved.

VBA example:

```
    Dim Modified As Boolean
    Modified = ThisDisplay.IsModified
    If Modified Then
        MsgBox "The display has been modified since the last save."
    Else
        MsgBox "The display has not been modified since the last save."
    End If
```

VB example:

```
    Dim Modified
    Modified = ThisDisplay.IsModified

    If Modified Then
        MsgBox "The display has been modified since the last save."
    Else
        MsgBox "The display has not been modified since” + _
          “ the last save."
    End If
```

Function IsPopupWindow

```
Function IsPopupWindow() As Boolean
```

TRUE if this display is running in a popup window or an embedded popup window; FALSE otherwise.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
Sub IsPopupWindow(o As GwxPick)
    'What is this coming from?
    Select Case (ThisDisplay.IsPopupWindow)
        Case True:
            MsgBox "It is a popup window."
        Case False:
            MsgBox "It is a parent window."
    End Select
End Sub
```

VB example:

```
    'What is this coming from?
    Select Case (ThisDisplay.IsPopupWindow)
        Case True:
            MsgBox "It is a popup window."
        Case False:
            MsgBox "It is a parent window."
    End Select
```

Function IsRuntimeMode

```
Function IsRuntimeMode() As Boolean
```

Returns TRUE is GraphWorX32 is currently in runtime mode, FALSE otherwise.

VBA example:

```
'Returns current mode.
Select Case (ThisDisplay.IsRuntimeMode)
    Case True:
        MsgBox "You are in runtime mode."
    Case False:
        MsgBox "You are in design mode."
End Select
```

VB example:

```
'Returns current mode.
Select Case (ThisDisplay.IsRuntimeMode)
    Case True:
        MsgBox "You are in runtime mode."
    Case False:
        MsgBox "You are in design mode."
End Select
```

Sub LogicalPointToClient

```
Sub LogicalPointToClient(x As Long, y As Long)
```

Converts GraphWorX32 logical coordinates to client coordinates.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
Dim obj As Object
Dim xL As Long
Dim yL As Long

Set obj = ThisDisplay.GetVisibleObjectFromName("ObjectX_gridL")
xL = Val(obj.text)
Set obj = ThisDisplay.GetVisibleObjectFromName("ObjectY_gridL")
yL = Val(obj.text)

Call ThisDisplay.LogicalPointToClient(xL, yL)

Set obj = ThisDisplay.GetVisibleObjectFromName("ObjectX_gridC")
obj.text = CStr(xL)
Set obj = ThisDisplay.GetVisibleObjectFromName("ObjectY_gridC")
obj.text = CStr(yL)
```

Sub LogicalPointToScreen

```
Sub LogicalPointToScreen(x As Long, y As Long)
```

Converts GraphWorX32 logical coordinates to screen coordinates.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
'We assume we have created a text object to store
'the data.

Dim obj As Object
Dim xL As Long
```

```
    Dim yL As Long

    Set obj = ThisDisplay.GetVisibleObjectFromName("ObjectX_gridL")
    xL = Val(obj.text)
    Set obj = ThisDisplay.GetVisibleObjectFromName("ObjectY_gridL")
    yL = Val(obj.text)

    Call ThisDisplay.LogicalPointToScreen(xL, yL)

    Set obj = ThisDisplay.GetVisibleObjectFromName("ObjectX_gridS")
    obj.text = CStr(xL)
    Set obj = ThisDisplay.GetVisibleObjectFromName("ObjectY_gridS")
    obj.text = CStr(yL)
```

Sub MaximizeWindow

```
Sub MaximizeWindow()
```

It maximizes the GraphWorX32 main window.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
ThisDisplay.MaximizeWindow
```

VB example:

```
ThisDisplay.MaximizeWindow
```

Sub MinimizeWindow

```
Sub MinimizeWindow()
```

Minimizes the GraphWorX32 main window.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
ThisDisplay.MinimizeWindow
```

VB example:

```
ThisDisplay.MinimizeWindow
```

Function MoveSelectionBackward

```
Function MoveSelectionBackward() As Boolean
```

Moves the selected object backward by one position. It is used to visually sort objects and to put one object below another.

VBA example:

```
    Dim obj As Object
    Dim obj2 As Object
    Set obj = ThisDisplay.GetVisibleObjectFromName("Object_red")
    Dim oRec As GwxEllipse
    Set oRec = obj

    ThisDisplay.DeselectAllObjects
    oRec.Selected = True
```

```
ThisDisplay.MoveSelectionBackward
```

Function MoveSelectionForward

```
Function MoveSelectionForward() As Boolean
```

Moves the selected object foreward by one position. It is used to visually sort objects and to put one object above another.

VBA example:

```
Dim obj As Object
Dim obj2 As Object
Set obj = ThisDisplay.GetVisibleObjectFromName("Object_red")
Dim oRec As GwxEllipse
Set oRec = obj

ThisDisplay.DeselectAllObjects
oRec.Selected = True

ThisDisplay.MoveSelectionForward
```

Function MoveSelectionToBack

```
Function MoveSelectionToBack() As Boolean
```

Move the selected object behind all of the other objects in the display.

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetVisibleObjectFromName("Object_red")
Dim oRec As GwxEllipse
Set oRec = obj

ThisDisplay.DeselectAllObjects
oRec.Selected = True

ThisDisplay.MoveSelectionToBack
```

Function MoveSelectionToFront

```
Function MoveSelectionToFront() As Boolean
```

Move the selected object in front of all of the other objects in the display

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetVisibleObjectFromName("Object_red")
Dim oRec As GwxEllipse
Set oRec = obj

ThisDisplay.DeselectAllObjects
oRec.Selected = True

ThisDisplay.MoveSelectionToFront
```

Function OpenEmbeddedWindow

```
Function OpenEmbeddedWindow(filename As String, center As Boolean, hidden As
Boolean) As Object
```

Opens a GraphWorX32 embedded window. Returns the object of the display loaded into the embedded window (of type **GwxDisplay**) on success, NULL on failure.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    Call ThisDisplay.OpenEmbeddedWindow("EmbeddedWindow.gdf", _
      True, False)
```

VB example:

```
    Call ThisDisplay.OpenEmbeddedWindow("EmbeddedWindow.gdf", _
      True, False)
```

Function OpenEmbeddedWinSetAliases

```
Function OpenEmbeddedWinSetAliases(filename As String, _
     center As Boolean, _
     hidden As Boolean, _
     aliasCommandType As GWXALIASCOMMAND, _
     aliasCommandString As String) As Object
```

Opens an embedded popup window with the specified display file and initially sets the specified aliases. If currently in runtime mode, the aliases will be set before any tags are requested. For descriptions of **aliasCommandType** and **aliasCommandString**, see **FileOpenSetAliases()**).

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    'Specify a text file. Take care all
    'of the white spaces (only TABS) dividing strings.
    'Set to use real tabs in your editor.
    'Template is:
    'aliasName1<TAB>aliasDef1
    'aliasName2<TAB>aliasDef2
    '<TAB> is the real TAB character.
    Dim strTextFileName As String
    strTextFileName = "AliasCommandString.txt"

    'The file contents:
    'SourceAlias1    100+abs({{gfwsim.sine.double}})*500
    'SourceAlias2    abs({{gfwsim.sine.double}})*1000
    'SourceAlias3    120+5*{{gfwsim.sine.double}}



    'Specify an embedded window:
    Dim strEmbdWndFileName As String
    strEmbdWndFileName = "EmbeddedWindowWithAliases.gdf"
    'To make this example working properly we
    'assume 3 process points in the display.
    'Put this code in "DataSource" in
    'Property Inspector in each of the process points:

    'ProcessPont N.1:
        'x= $"SourceAlias1: The value is "$ + ...
        '... tostring(<<SourceAlias1>>) + $" units."$
```

```
    'ProcessPont N.2:
        'x= $"SourceAlias2: The value is "$ + ...
        '... tostring(<<SourceAlias2>>) + $" units."$
    'ProcessPont N.3:
        'x= $"SourceAlias3: The value is "$ + ...
        '... tostring(<<SourceAlias3>>) + $" units."$

    'Adjust properties according to the string lenght and format.
    'See "Creating Expressions Using the Expression Editor"
    'in the help.


    Call ThisDisplay.OpenEmbeddedWinSetAliases(strEmbdWndFileName, _
                                               True, _
                                               False, _
                                               AliasSetFromFile, _
                                               strTextFileName)
```

VB example:

```
    'Specify a text file. Take care all
    'of the white spaces (only TABS) dividing strings.
    'Set to use real tabs in your editor.
    'Template is:
    'aliasName1<TAB>aliasDef1
    'aliasName2<TAB>aliasDef2
    '<TAB> is the real TAB character.
    Dim strTextFileName
    strTextFileName = "AliasCommandString.txt"

    'The file contents:
    'SourceAlias1    100+abs({{gfwsim.sine.double}})*500
    'SourceAlias2    abs({{gfwsim.sine.double}})*1000
    'SourceAlias3    120+5*{{gfwsim.sine.double}}



    'Specify an embedded window:
    Dim strEmbdWndFileName
    strEmbdWndFileName = "EmbeddedWindowWithAliases.gdf"
    'To make this example working properly we
    'assume 3 process points in the display.
    'Put this code in "DataSource" in
    'Property Inspector in each of the process points:

    'ProcessPont N.1:
        'x= $"SourceAlias1: The value is "$ + ...
        '... tostring(<<SourceAlias1>>) + $" units."$
    'ProcessPont N.2:
        'x= $"SourceAlias2: The value is "$ + ...
        '... tostring(<<SourceAlias2>>) + $" units."$
    'ProcessPont N.3:
        'x= $"SourceAlias3: The value is "$ + ...
        '... tostring(<<SourceAlias3>>) + $" units."$

    'Adjust properties according to the string lenght and format.
    'See "Creating Expressions Using the Expression Editor"
    'in the help.


    Call ThisDisplay.OpenEmbeddedWinSetAliases(strEmbdWndFileName, _
                                               True, _
                                               False, _
```

```
                                                    AliasSetFromFile, _
                                                    strTextFileName)
```

Function OpenPopupWindow

```
Function OpenPopupWindow(filename As String, _
     modal As Boolean, _
     center As Boolean, _
     hidden As Boolean) As Object
```

Opens a GraphWorX32 popup window. Returns the object of the display loaded into the popup window (of type **GwxDisplay**) on success, NULL on failure.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
Sub OpenPopupWnd(o As GwxPick)
    'Open display "PopupWindow.gdf":
    Call ThisDisplay.OpenPopupWindow("PopupWindow.gdf", False, _
      True, False)
End Sub
```

VBA example:

```
'Open "PopupWindow.gdf":
    Call ThisDisplay.OpenPopupWindow("PopupWindow.gdf", False, _
       True, False)
```

Function OpenPopupWinSetAliases

```
Function OpenPopupWinSetAliases(filename As String, _
     modal As Boolean, _
     center As Boolean, _
     hidden As Boolean, _
     aliasCommandType As GWXALIASCOMMAND, _
     aliasCommandString As String) As Object
```

Opens a popup window with the specified display file and initially sets the specified aliases. If currently in runtime mode, the aliases will be set before any tags are requested. For descriptions of **aliasCommandType** and **aliasCommandString**, see **FileOpenSetAliases()**).

Note: Not available in ActiveX ToolWorX.

Function OpenSetAliasesDialog

```
Function OpenSetAliasesDialog(aliasCommandType As GWXALIASCOMMAND, _
     aliasCommandString As String, _
     objectName As String) As Boolean
```

Opens the Set Aliases Dialog. This dialog is used to allow the operator to enter new alias definitions for a set of specified alias names. For descriptions of **aliasCommandType** and **aliasCommandString**, see **FileOpenSetAliases()**). The **aliasCommandString** parameter is the primary means of configuring the dialog. For each unique alias name specified in **aliasCommandString**, there will be a tab added to the dialog box. Alias definitions specified with the alias names will be listed in the appropriate property page's combo box. The **objectName** parameter specifies which visible object the new alias settings will apply to (if no object name is specified, new alias settings will be applied to the entire display).

Note: Not available in ActiveX ToolWorX.

VBA example:

```
'1. AliasSetFromFile

'Specify a text file. Take care all
'of the white spaces (only TABS) dividing strings.
'Set to use real tabs in your editor.
'Template is:
'aliasName1<TAB>aliasDef1
'aliasName2<TAB>aliasDef2
'<TAB> is the real TAB character.
Dim strTextFileName As String
strTextFileName = "AliasCommandString.txt"

'The file contents:
'SourceAlias1    100+abs({{gfwsim.sine.double}})*500
'SourceAlias2    abs({{gfwsim.sine.double}})*1000
'SourceAlias3    120+5*{{gfwsim.sine.double}}

MsgBox "AliasSetFromFile"

Call ThisDisplay.OpenSetAliasesDialog(AliasSetFromFile, _
                                      strTextFileName, _
                                      "ProcessPoint")

'2.
'Compose a command string:
Dim strAliasCommand As String
strAliasCommand = "SourceAlias1" + Chr(9) + _
                    "100+abs({{gfwsim.sine.double}})*500" + _
                  Chr(13) + _
                  "SourceAlias1" + Chr(9) + _
                    "abs({{gfwsim.sine.double}})*1000" + _
                  Chr(13) + _
                  "SourceAlias2" + Chr(9) + _
                    "120+5*{{gfwsim.sine.double}}"


Call ThisDisplay.OpenSetAliasesDialog(AliasSetDirect, _
                                      strAliasCommand, _
                                      "ProcessPoint")
```

Function OpenTagBrowser, OpenTagBrowser2

```
Function OpenTagBrowser(hWndParent As Long, _
     tagName As String) As Boolean
```

Function opens the Tag Browser with the specified window handle as the parent window. The parameter **tagName** receives the tag name selected by the user. Returns TRUE if the user hit the OK button, FALSE if the user hit the Cancel button.

VBA example:

```
Dim handle As Long
handle = ThisDisplay.GetFrameWindowHandle

'Use the handle to specify the parent window.
Dim tagName As String
Call ThisDisplay.OpenTagBrowser(handle, tagName)
```

VB example:

```
Dim handle
```

```
handle = ThisDisplay.GetFrameWindowHandle

'Use the handle to specify the parent window.
Dim tagName
Call ThisDisplay.OpenTagBrowser2(handle, tagName)
```

Function PopAllCurrentSymbol

```
Function PopAllCurrentSymbol() As Boolean
```

Pops the current symbol edit level all the way back to the root level. Returns TRUE for success, FALSE for failure.

Note: See also **GwxSymbol::PushCurrentSymbol()** . *Available only in configure mode.*

VBA example

```
ThisDisplay.PopAllCurrentSymbol
```

Function PopCurrentSymbol

```
Function PopCurrentSymbol() As Boolean
```

Pops the current symbol edit level up one level. Returns TRUE for success, FALSE for failure. (See also **GwxSymbol::PushCurrentSymbol()**).

Note: Available only in configure mode.

Function QueryDataType

```
Function QueryDataType(dataSource As String, dataType As GWXDATATYPE) As
Boolean
```

Queries the OPC server for the data type of the specified tag name (**dataSource**). See **GwxPoint** for possible values of GWXDATATYPE.

VBA example:

```
Dim strSource As String
strSource = "ICONICS.Simulator.1\SimulatePLC.Sine"

Dim lDataType As GWXDATATYPE
Call ThisDisplay.QueryDataType(strSource, _
                               lDataType)

Dim strDataType As String

strDataType = "DataType"
Select Case lDataType
    Case 0
        strDataType = strDataType + "Empty"
    Case 2
        strDataType = strDataType + "Short"
    Case 3
        strDataType = strDataType + "Long"
    Case 4
        strDataType = strDataType + "Float"
    Case 5
        strDataType = strDataType + "Double"
    Case 8
        strDataType = strDataType + "String"
    Case 11
        strDataType = strDataType + "Bool"
    Case 16
        strDataType = strDataType + "Char"
```

```
        Case 17
            strDataType = strDataType + "Byte"
        Case 18
            strDataType = strDataType + "Word"
        Case 19
            strDataType = strDataType + "Dword"
        Case Else
            strDataType = "Type undefined."
    End Select


    MsgBox "Data type of the SimulatePLC.Sine source is: " + _
            CStr(strDataType)
```

VB example:

```
    Dim strSource
    strSource = "ICONICS.Simulator.1\SimulatePLC.Sine"

    Dim lDataType
    Call ThisDisplay.QueryDataType2(strSource, _
                                    lDataType)

    Dim strDataType

    strDataType = "DataType"
    Select Case lDataType
        Case 0
            strDataType = strDataType + "Empty"
        Case 2
            strDataType = strDataType + "Short"
        Case 3
            strDataType = strDataType + "Long"
        Case 4
            strDataType = strDataType + "Float"
        Case 5
            strDataType = strDataType + "Double"
        Case 8
            strDataType = strDataType + "String"
        Case 11
            strDataType = strDataType + "Bool"
        Case 16
            strDataType = strDataType + "Char"
        Case 17
            strDataType = strDataType + "Byte"
        Case 18
            strDataType = strDataType + "Word"
        Case 19
            strDataType = strDataType + "Dword"
        Case Else
            strDataType = "Type undefined."
    End Select


    MsgBox "Data type of the SimulatePLC.Sine source is: " + _
            CStr(strDataType)
```

Function QueryRanges

```
Function QueryRanges(dataSource As String, _
     lowRange As Double, _
```

```
      highRange As Double) As Boolean
```

Queries the OPC server for the high and low range values of the specified tag name (**dataSource**).

VBA example:

```
Dim strSource As String
strSource = "ICONICS.Simulator.1\SimulatePLC.Sine"

Dim lowRange As Double
Dim highrange As Double

If ThisDisplay.QueryRanges(strSource, _
                           lowRange, _
                           highrange) _
                           Then
   MsgBox "The High Range is: " + CStr(highrange) + _
       Chr(13) + "The Low Range is: " + CStr(lowRange)
End If
```

VB example:

```
Dim strSource
strSource = "ICONICS.Simulator.1\SimulatePLC.Sine"

Dim lowRange
Dim highrange

If ThisDisplay.QueryRanges2(strSource, _
                           lowRange, _
                           highrange) _
                           Then
   MsgBox "The High Range is: " + CStr(highrange) + _
       Chr(13) + "The Low Range is: " + CStr(lowRange)
End If
```

Sub RefreshWindow

```
Sub RefreshWindow()
```

Redraws the visible portion of a GraphWorX32 display.

VBA example:

```
'To run this script you need to create an object with _
'name "Test2".

'Toggle Redraw so the display will
'not be able to automatically
'refresh. At the end of this script
'we will manualy call RefreshWindow.
ThisDisplay.Redraw = False

'Get rectangle from name:
Dim oRec As GwxRectangle
Set oRec = ThisDisplay.GetVisibleObjectFromName("Test2")
oRec.Selected = True

'Set Redraw to True:
ThisDisplay.Redraw = True

'and finally use refresh:
```

```
Call ThisDisplay.RefreshWindow
```

VB example:

```
'This routine changes the dimensions of display with following _
'display recovery.


'Declare coordinates as long values
Dim widthDisplay
Dim heightDisplay

Dim widthDisplayMem
Dim heightDisplayMem

'Remember current dimensions:
'TO DO!
Call ThisDisplay.GetDisplayDimensions2(widthDisplayMem, _
   heightDisplayMem)
widthDisplayMem = 1036
heightDisplayMem = 767

'Set up new parameters:
widthDisplay = 600
heightDisplay = 400

'Set the display parameters:
Call ThisDisplay.SetDisplayDimensions(widthDisplay, _
    heightDisplay)
'and refresh the window:
ThisDisplay.RefreshWindow

MsgBox "When you click on OK the dimensions will be restored."

'Again, set display parameters:
Call ThisDisplay.SetDisplayDimensions(widthDisplayMem, _
   heightDisplayMem)

'and once more refresh the window:
ThisDisplay.RefreshWindow
```

Sub RemoveLayer

```
Sub RemoveLayer(layerName As String)
```

Removes the specified layer from the display.

Function ReplaceFilePath

```
Function ReplaceFilePath(OldSubstring As String, _
     NewSubsring As String) As Long
```

Works over path name parameters of pick dynamics. Returns 0 on success and HRESULT when something fails.

VBA example:

```
'replaces file path in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceFilePath("C:\Windows\Temp", _
  "D:\Temp")
'if Status <> 0 then there was no
'replacement performed or an error occurs
```

```
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

Function ReplaceHost

```
Function ReplaceHost(OldHostName As String, _
     NewHostName As String) As Long
```

Works over both data source tags and path name attributes and replaces node name **substring** within URL path only. It returns 0 on success and HRESULT when something fails.

Function ReplaceHostEx

```
Function ReplaceHostEx(OldHostNameSubstring As String, _
     NewHostNameSubstring As String, _
     MatchCase As Boolean, _
     MatchWholeWord As Boolean) As Long
```

Works over both data source tags and path name attributes and replaces node name **substring** within URL path only, and supports case-sensitivity, wildcard strings and MatchWholeWord flag. It returns 0 on success and HRESULT when something fails.

Function ReplaceTag

```
Function ReplaceTag(OldSubstring As String, _
     NewSubstring As String) As Long
```

Works over data source tags in dynamic actions of all display objects. Replaces **OldSubstring** with **NewSubstring** in certain situations. Returns 0 on success and HRESULT when something fails.

VBA example:

```
'VBA example, works only in configure mode
'replaces tag name in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceTag("gfwsim.ramp.float", "gfwsim.sine.double")
'if Status <> 0 then there was no replacement
'performed or an error occurs

If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

long ReplaceStringInString

```
long ReplaceStringInString (BSTR StringToReplace, _
     BSTR OldSubstring, _
     BSTR NewSubsring)
```

Replaces **OldSubstring** with **NewSubstring** inside (within) StringToReplace. Returns 0 on success and HRESULT when something fails.

Sub RestoreWindow

```
Sub RestoreWindow()
```

Restores the (non-minimized/non-maximized) window size and position.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
ThisDisplay.RestoreWindow
```

VB example:

```
ThisDisplay.RestoreWindow
```

Function SelectPaletteColor, SelectPaletteColor2

```
Function SelectPaletteColor(SelectedColor As OLE_COLOR) As Boolean
```

Opens the GraphWorX32 color palette dialog box. The parameter **SelectedColor** receives the color selected by the user. Returns TRUE if the user hit the OK button, FALSE if the user hit the Cancel button.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
    Dim color As OLE_COLOR
    Call ThisDisplay.SelectPaletteColor(color)
    MsgBox "OLE color is defined as: " + CStr(color)
```

VB example:

```
    Dim color
    Call ThisDisplay.SelectPaletteColor2(color)

    MsgBox "OLE color is defined as: " + CStr(color)
```

Function SelectPaletteColorRGB

```
Function SelectPaletteColorRGB(red, green, blue) As Boolean
```

Pops up a dialog with a color palette and allows you to select one color between them.

Returns TRUE if a color is selected; returns FALSE in all other cases.

Each color displayed in the computer monitor can be decomposed into 3 components : red, green and blue. Each of these component colors can have an intensity that ranges from 0 (the component is not present) to 255 (the maximum that each component color can contribute to a color).

If the user selects a color, the 3 parameters are filled with the red, green and blue composition of the selected color. For example, if you select a bright yellow color, then you will get back the following:

Red= 255
Green= 255
Blue= 0

Note: This method works in Visual Basic for Applications only if the parameters are declared as Variant.

VBA example:

```
    Dim lr As Variant
    Dim lg As Variant
    Dim lb As Variant
```

```
r = 0
g = 255
b = 255

Call ThisDisplay.SelectPaletteColorRGB(lr, lg, lb)

MsgBox "RGB color is defined as: " + _
        " " + CStr(lr) + _
        " " + CStr(lg) + _
        " " + CStr(lb)
```

VB example:

```
Dim r
Dim g
Dim b

r = 0
g = 255
b = 255
Call ThisDisplay.SelectPaletteColorRGB(r, g, b)

MsgBox "RGB color is defined as " + _
        CStr(r) + " " + CStr(g) + " " + CStr(b)
```

Function SetAliasDefinition

```
Function SetAliasDefinition(aliasName As String, _
     newDefinition As String) As Boolean
```

Sets the alias definition of the specified alias name for all dynamic objects in the display. This function can be used in runtime mode to easily change the data connections of dynamic objects on-the-fly. Returns FALSE if no matching alias names were found, TRUE otherwise.

Note: See also GwxVisible::SetAliasDefinition and GwxDynamic::SetAliasDefinition.

VBA example:

```
Dim strAliasName As String
strAliasName = "SourceAlias1"

Call ThisDisplay.setAliasDefinition(strAliasName, _
    "{{gfwsim.cosine.double}}*20")
MsgBox "The new definition for the alias" + _
    " ""SourceAlias1"" is {{gfwsim.cosine.double}}*20."
```

Function SetAliases

```
Function SetAliases(commandString As String) As Boolean
```

Sets multiple alias definitions for the entire display. The **commandString** parameter has the following format:

aliasName1<TAB>aliasDef1<CR-LF>aliasName2<TAB>aliasDef2<CR-LF> … etc.

VBA example:

```
'Compose a command string:
'See GwxDisplay::SetAliasDefinition

Dim strAliasCommand As String
```

```
strAliasCommand = "SourceAlias1" + Chr(9) + _
                  "100+abs({{gfwsim.sine.double}})*500"
'Chr(13) is dividing each command line.

MsgBox "Now we define another source for the SourceAlias1: " + _
    "100+abs({{gfwsim.sine.double}})*500"
Call ThisDisplay.SetAliases(strAliasCommand)
```

Function SetAliasesFromFile

```
Function SetAliasesFromFile(filename As String) As Boolean
```

Sets multiple alias definitions for the entire display. The **filename** parameter specifies a text file containing tab-separated pairs of alias names and alias definitions (the format of this file is the same format as the string passed to **GwxDisplay**'s **SetAliases()** method). Files of this type can be created in **Notepad**, can be exported from **Excel** (save as Tab Delimited), and can be created using **Gwx32's Alias File Editor**.

VBA example:

```
Dim strTextFileName As String

'Specify a text file. Take care all
'of the white spaces (only TABS) dividing the strings.
'Set to use real tabs in your editor.
'Template is:
'aliasName1<TAB>aliasDef1<CR-LF>
'aliasName2<TAB>aliasDef2<CR-LF>
'....
'..
'.
strTextFileName = "AliasCommandString.txt"

MsgBox "Now we define another source for the Local alias." + _
    "See the value in the top right hand corner of the display."

Call ThisDisplay.SetAliasesFromFile(strTextFileName)
```

VB example:

```
Dim strTextFileName

'Specify a text file. Take care all
'of the white spaces (only TABS) dividing the strings.
'Set to use real tabs in your editor.
'Template is:
'aliasName1<TAB>aliasDef1<CR-LF>
'aliasName2<TAB>aliasDef2<CR-LF>
'....
'..
'.
strTextFileName = "AliasCommandString.txt"

MsgBox "Now we define another source for the Local alias." + _
    "See the value in the top right hand corner of the display."

Call ThisDisplay.SetAliasesFromFile(strTextFileName)
```

Sub SetDisplayDimensions

```
Sub SetDisplayDimensions(width As Long, height As Long)
```

Sets the display dimensions (work area/world bounds) of the currently loaded display.

VBA example:

```
Sub SetDisplayDimensions(o As GwxPick)
    'Declare coordinates as long values
    Dim widthDisplay As Long
    Dim heightDisplay As Long

    Dim widthDisplayMem As Long
    Dim heightDisplayMem As Long

    'Remember current dimensions:
    Call ThisDisplay.GetDisplayDimensions(widthDisplayMem, _
       heightDisplayMem)

    'Set new values:
    widthDisplay = 600
    heightDisplay = 1000

    'Set display parameters:
    Call ThisDisplay.SetDisplayDimensions(widthDisplay, _
      heightDisplay)
    'and refresh the window:
    ThisDisplay.RefreshWindow

    MsgBox "When you click on OK the dimensions" + _
        " will be restored." + _
        Chr(13) + " (width = " + _
        Conversion.CStr(widthDisplayMem) + _
        " and height = " + _
        Conversion.CStr(heightDisplayMem) + _
        ")"

    'Again, set display parameters:
    Call ThisDisplay.SetDisplayDimensions(widthDisplayMem, _
      heightDisplayMem)
    'and once more refresh the window:
    ThisDisplay.RefreshWindow
End Sub
```

VB example:

```
'Declare coordinates as long values
    Dim widthDisplay
    Dim heightDisplay

    Dim widthDisplayMem
    Dim heightDisplayMem

    'Remember current dimensions:
    'TO DO!
    Call ThisDisplay.GetDisplayDimensions2(widthDisplayMem, _
        heightDisplayMem)
    widthDisplayMem = 1036
    heightDisplayMem = 767

    'Set up new values:
    widthDisplay = 600
    heightDisplay = 400

    'Set display parameters:
```

```
    Call ThisDisplay.SetDisplayDimensions(widthDisplay, _
       heightDisplay)
    'and refresh the window:
    ThisDisplay.RefreshWindow

    MsgBox "When you click on OK the dimensions will be restored."

    'Again, set display parameters:
    Call ThisDisplay.SetDisplayDimensions(widthDisplayMem, _
       heightDisplayMem)

   'and once more refresh the window:
    ThisDisplay.RefreshWindow
```

Sub SetEventScriptText

```
Sub SetEventScriptText(Event As GWXEVENTNAME, text As String)
```

GraphWorX32 allows for associating VBScript and JScript with events. The scripts associated with an event are executed when the event is triggered (for example, after runtime starts, or before animating a layer). These scripts can be edited through the new integrated script toolbar or through the OLE automation interface. Each script can be written using VBScript or JScript language independently from the other event. This function allows for writing (changing) the script source code currently selected for the specified event.

Sub SetEventScriptType

```
Sub SetEventScriptType(Event As GWXEVENTNAME, Type As GWXSCRIPTTYPE)
```

GraphWorX32 allows for associating VBScript and JScript with events. The scripts associated with an event are executed when the event is triggered (for example, after runtime starts, or before animating a layer). These scripts can be edited through the new integrated script toolbar or through the OLE automation interface. Each script can be written using VBScript or JScript language independently from the other event. This function allows for writing (changing) the script source code currently associated with the specified event.

VBA example:

```
'Works in configure and runtime mode
'Creates VB script on event PostRuntimeStart

ThisDisplay.SetEventScriptType _
PostRuntimeStart, ScriptTypeVBScript

ThisDisplay.SetEventScriptText _
PostRuntimeStart, "MsgBox " & Chr(34) & _
"Test OK" & Chr(34)
```

Note: Correpsonding VBScript Example:
ThisDisplay.SetEventScriptText 23,"msgbox ""test"" "
ThisDisplay.SetEventScriptType 23,1

Function SetFileVersion

```
Function SetFileVersion(NewFileVersion As GWXSETFILEVERSION) _
     As GWXGETFILEVERSION
```

The function **SetFileVersion()** sets a display version for the next save. Note that the set of versions supported for saving is less than the set of versions detected on reading the file. In other words, GraphWorX32 detects more versions of files, but it allows saving only in the more

recent versions; version 5.20 and older versions are not supported. If there is an attempt to save in an unsupported older version, then GraphWorX32 saves in the version `SetVer600`.

```
SetFileVersion() input values
2  .. SetVer600SP
3  .. SetVerMGraphics40
4  .. SetVer610
9999  .. SetVerLatest (&H270F)
```

The automation interface is very powerful, and it can be used for automated display processing (e.g. there is a need to upgrade or downgrade a group of displays to a certain version, or to make certain changes in the displays while preserving the existing version).

VBA example:

```
    'Keep the latest version.
    MsgBox "Keep the latest version."
    ThisDisplay.SetFileVersion (SetVerLatest)
```

VB example:

```
    'Keep the latest version.
    MsgBox "Keep the latest version."
    ThisDisplay.SetFileVersion (SetVerLatest)
```

VB Code Sample:

```
Private Sub DisplayVersion()
    ' Specify the path and filename of an existing display
    Const strPath As String = _
"C:\PathToYourDisplay\"
    Const strFilename As String = _
strPath & "YourDisplay.gdf"
    Const SetVer600SP as Long = 2
    Const VerLatest as Long = 9999

    ' Create GWX instance
    Dim g As New Gwx32.GwxDisplay

    ' Check if GWX was created
    If g Is Nothing Then
        MsgBox "Failed to create GWX"
        Exit Sub
    End If

    ' Show GWX in the front and load the requested display
    Call g.ShowWindow
    Call g.BringWindowToTop
    Call g.FileNew
    Call g.FileOpen(strFilename)

    ' Check the file version
    Dim ver As long
    ver = GetFileVersion(g)

    ' If this is the latest version, save in the
    ' version 6.00
    If (ver = VerLatest) Then
        g.SetFileVersion(SetVer600SP)
        g.FileSave
```

```
    End If

    ' Exit GWX, the FileNew suppresses the message box
    ' if to save changes.
    g.FileNew
    g.ExitApplication
End Sub

' Get file version and trace the current version in the
' Immediate window for debugging purposes.
Private Function GetFileVersion(g As GwxDisplay) As Long
    Const OpenFileError as Long = -1
    Const Ver520 as Long = 0
    Const Ver600 as Long = 2
    Const Ver600SP as Long = 2
    Const VerMGraphics40 as Long = 2
    Const Ver610 as Long = 2
    Const VerLatest as Long = 9999

    Dim ver As long

    ' Check the version
    ver = g.GetFileVersion
    Select Case (ver)
        Case OpenFileError:
            Debug.Print "Version: FileOpenError"
        Case Ver520:
            Debug.Print "Version: Ver520"
        Case Ver600:
            Debug.Print "Version: Ver600"
        Case Ver600SP:
            Debug.Print "Version: Ver600SP"
        Case VerMGraphics40:
            Debug.Print "Version: VerMGraphics40"
        Case Ver610:
            Debug.Print "Version: Ver610"
        Case VerLatest:
            Debug.Print "Version: VerLatest"
    End Select

    ' Check if the display contains VBA
    If g.GetFileContainsVBA Then
        Debug.Print "VBA: Yes"
    Else
        Debug.Print "VBA: No"
    End If

    GetFileVersion = ver
End Function
```

Function SetGASThemes

```
Function SetGASThemes(Themes As String) As Boolean
```

Sets Global Alias themes. The format is the same as generated by the Theme Browser, so it is recommended to use the Theme Browser to generate the desired combination and copy/paste the result into this function as string.

VBA, VB example:

```
'VBA/VB Script example, works only in runtime mode
'example for setting and getting themes (Floors theme is changed in child
windows)
```

```
ThisDisplay.SetGASThemes "Buildings=BuildingB;Floors=Floor2/+1;"
MsgBox ThisDisplay.GetGASThemes
```

Function SetGradientFill

```
Function SetGradientFill(style As GWXGRADIENTSTYLE, _
     color1 As OLE_COLOR, _
     color2 As OLE_COLOR, _
     isTwoColor As Boolean, _
     brightness As Single, _
     reverseColors As Boolean, _
     steps As Integer, _
     horizontalOffset As Single, _
     verticalOffset As Single) As Boolean
```

Sets a gradient fill for display background.

The parameter **style** can be one of the following values:

GradientHorizontal = 0
GradientVertical = 1
GradientSquare = 2

The rest of the parameters correspond to those in the gradient fill configuration user interface.

Function SetHeadObject

```
Function SetHeadObject(objectName As String) As Boolean
```

Sets the head object to the visible object with the specified name. Returns TRUE for success, FALSE for failure.

VBA example:

```
     'If you want to select an object with the name "Object"
     ThisDisplay.SetHeadObject ("Object")
```

Function SetLayerOverrideHide

```
Function SetLayerOverrideHide(layerName As String, _
hidden As Boolean) As Boolean
```

Sets the layer override hide property of the specified display layer.

Sub SetModifiedFlag

```
Sub SetModifiedFlag(Modified As Boolean)
```

The modified flag is used to determine if the display has been modified since the laste time it was saved. If the modified flag is set to true, then GraphWorX32 will request to save the changes to the display before closing the display or loading a new display.

VBA example:

```
    ThisDisplay.SetModifiedFlag (True)
    MsgBox "The modified flag has been set to ""True""."
```

Sub SetPrintWhiteBackground

```
Sub SetPrintWhiteBackground(WhiteBackground As Boolean, PrintArea As Long)
```

Sets the view area and the white background color for printing.

VBA example:

```
'Set an appropriate display layout:
'PV_NOCHANGE = 0,
'PV_HOME = 1,
'PV_FITTOWINDOW = 2,
'PV_WORLD = 3
MsgBox "The display layout is: PV_NOCHANGE"
Call ThisDisplay.SetPrintWhiteBackground(True, 0)
```

VB example:

```
Call ThisDisplay.SetPrintWhiteBackground(True, 0)

MsgBox "The white background is set to True."
```

Function SetRuntimeFocusProperties

```
Function SetRuntimeFocusProperties(ShowHandCursor As Boolean, _
     ShowFocusRectangle As Boolean, _
     fillColor As OLE_COLOR, _
     BorderColor As OLE_COLOR) As Boolean
```

Sets the attributes for runtime focus highlights.

Function SetRuntimeUpdateRate

```
Function SetRuntimeUpdateRate(updateRate As Long) As Boolean
```

Sets the OPC data update rate for the display.

Function SetSystemWideLanguage

```
Function SetSystemWideLanguage(Language As String, _
     LanguageSubset As String, _
     LocaleID As Long, _
     SynchronizeLCID As Boolean) As Boolean
```

Function sets the GENESIS language and language subset for the document, the LocaleID for application, and the flag if to synchronize document language with an application.

Language	ID
Czech	1029
Dutch	2067
English	1033
French	1036
German	1031
Italian	1040
Russian	1049
Spanish	1034

VBA example:

```
Dim strLanguage As String
Dim strLanguageSubSet As String
Dim lLocaleID As Long
Dim bSynchronizeLCID As Boolean

strLanguage = "Czech"
strLanguageSubSet = "Czech.Test"
lLocaleID = 1029
bSynchronizeLCID = False
```

```
    Call ThisDisplay.SetSystemWideLanguage( _
                                    strLanguage, _
                                    strLanguageSubSet, _
                                    lLocaleID, _
                                    bSynchronizeLCID)
```

Function SetTooltipProperties

```
Function SetTooltipProperties(ShowDynamicTips As Boolean, _
     ShowDynamicObjectName As Boolean, _
     ShowDynamicObjectDescription As Boolean, _
     ShowDataSourceName As Boolean, _
     ShowDataSourceValue As Boolean, _
     ShowStaticTooltips As Boolean, _
     ShowStaticObjectName As Boolean, _
     ShowStaticObjectDescription As Boolean) As Boolean
```

Sets the tooltip properties for the display.

Function SetTransparency

```
Function SetTransparency(EnableTransparency As Boolean, _
     transparentColor As OLE_COLOR, _
     EnableAlpha As Boolean, _
     AlphaBlending As Integer) As Boolean
```

Sets display transparency in runtime. Works only in GraphWorX32 on Windows 2000 and higher.

VBA example:

```
'Works in runtime mode
'this example toggles transparency and transluency settings

Dim  TE As Boolean, TC As OLE_COLOR, AE As Boolean, _
AB As Integer
ThisDisplay.GetTransparency TE, TC, AE, AB

'following lines enable/disable transparency and transluency
If TE Then TE = False Else TE = True
If AE Then AE = False Else AE = True

'if transluency is enabled, sets alpha blending value to 100 of 255
If AE Then AB = 100 Else AB = 0

'applies changed values
ThisDisplay.SetTransparency TE, TC, AE, AB
MsgBox "Transparency: " & TE & vbCrLf & "Color: " _
& TC & vbCrLf & "Transluency: " & _
AE & vbCrLf & "Color: " & AB
```

Sub SetVCRMode

```
Sub SetVCRMode(Enable As Boolean)
```

Sets the state of VCR mode. It indicates whether the data can be replayed from the HDA database or not.

VBA example:

```
    If Not ThisDisplay.GetVCRMode Then
        ThisDisplay.SetVCRMode (True)
```

```
    End If
```

VB example:

```
    If Not ThisDisplay.GetVCRMode Then
        ThisDisplay.SetVCRMode (True)
    End If
```

Sub SetViewDimensions

```
Sub SetViewDimensions(left As Long, _
     top As Long, _
     width As Long, _
     height As Long)
```

Sets the GraphWorX32 view rectangle's size and location. The view dimensions define what portion of the work area/whole display is visible. Setting the view dimensions can be used to zoom and pan the view of a display.

VBA example:

```
Sub SetViewDimensions(o As GwxPick)
    'Declare coordinates as long values
    Dim viewLeft As Long
    Dim viewTop As Long
    Dim viewWidth As Long
    Dim viewHeight As Long

    'Backup values
    Dim bckLeft As Long
    Dim bckTop As Long
    Dim bckWidth As Long
    Dim bckHeight As Long

    'Remember current dimensions:
    Call ThisDisplay.GetViewDimensions(bckLeft, _
                                       bckTop, _
                                       bckWidth, _
                                       bckHeight)

    'Set new dimensions:
    viewLeft = 0
    viewTop = 0
    viewWidth = 1036
    viewHeight = 767

    'Set the view parameters:
    Call ThisDisplay.SetViewDimensions(viewLeft, _
                                       viewTop, _
                                       viewWidth, _
                                       viewHeight)
    'and refresh the window:
    ThisDisplay.RefreshWindow

    MsgBox "When you click on OK the view dimensions" + _
        " will be restored to the previous ones." + _
        Chr(13) + " (Left = " + _
        Conversion.CStr(bckLeft) + _
        " , Top = " + _
```

```
        Conversion.CStr(bckTop) + _
        " , Width = " + _
        Conversion.CStr(bckWidth) + _
        " and Height = " + _
        Conversion.CStr(bckHeight) + _
        ")"

    'Again, set the view parameters:
    Call ThisDisplay.SetViewDimensions(bckLeft, _
                                       bckTop, _
                                       bckWidth, _
                                       bckHeight)

    'and once more refresh the window:
    ThisDisplay.RefreshWindow
End Sub
```

VB example:

```
    'Declare coordinates as long values
    Dim viewLeft
    Dim viewTop
    Dim viewWidth
    Dim viewHeight
    'Backup values
    Dim bckLeft
    Dim bckTop
    Dim bckWidth
    Dim bckHeight

    'Remember current dimensions:
    'TO DO!
    Call ThisDisplay.GetViewDimensions2(bckLeft, _
                                       bckTop, _
                                       bckWidth, _
                                       bckHeight)

    bckLeft = 0
    bckTop = 0
    bckWidth = 1131
    bckHeight = 767


    'New dimensions: <0, 0, 502, 341>
    viewLeft = 0
    viewTop = 0
    viewWidth = 502
    viewHeight = 341

    'Set the view parameters:
    Call ThisDisplay.SetViewDimensions(viewLeft, _
                                       viewTop, _
                                       viewWidth, _
                                       viewHeight)
    'and refresh the window:
    ThisDisplay.RefreshWindow

    MsgBox "When you click on OK the view dimensions" + _
        " will be restored to the previous ones."


    'Again, set the view parameters:
    Call ThisDisplay.SetViewDimensions(bckLeft, _
```

```
                                        bckTop, _
                                        bckWidth, _
                                        bckHeight)

    'and once more refresh the window:
    ThisDisplay.RefreshWindow
```

Sub SetWindowDimensionsPercent

```
Sub SetWindowDimensionsPercent(left As Single, _
     top As Single, _
     width As Single, _
     height As Single)
```

Sets the GraphWorX32 main window size and location as a percentage of the total screen size. Parameters should be values in the range of 0.0 to 1.0.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
Sub SetWndDimPercent(o As GwxPick)
    'Declare coordinates as Single values
    Dim viewLeft As Single
    Dim viewTop As Single
    Dim viewWidth As Single
    Dim viewHeight As Single

    'Set new values:
    viewLeft = -0.002
    viewTop = -0.0025
    viewWidth = 0.45
    viewHeight = 0.5

    'Set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPercent(viewLeft, _
                                                viewTop, _
                                                viewWidth, _
                                                viewHeight)
    'and refresh the window:
    ThisDisplay.RefreshWindow

    MsgBox "When you click on OK the view dimensions" + _
        " will be restored to the previous ones."

    'Again, set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPercent(1 / viewLeft, _
                                                1 / viewTop, _
                                                1 / viewWidth, _
                                                1 / viewHeight)

    'and once more refresh the window:
    ThisDisplay.RefreshWindow
    'Rise window from the task bar
    ThisDisplay.MaximizeWindow

End Sub
```

VB example:

```
    'Declare coordinates
    Dim viewLeft
    Dim viewTop
    Dim viewWidth
    Dim viewHeight

    'Set new dimensions:
    viewLeft = -0.002
    viewTop = -0.0025
    viewWidth = 0.5
    viewHeight = 0.45

    'Set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPercent(viewLeft, _
                                                viewTop, _
                                                viewWidth, _
                                                viewHeight)
    'and refresh the window:
    ThisDisplay.RefreshWindow

    MsgBox "When you click on OK the view dimensions" + _
        "will be restored to the previous ones."

    'Again, set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPercent(1 / viewLeft, _
                                                1 / viewTop, _
                                                1 / viewWidth, _
                                                1 / viewHeight)

    'and once more refresh the window:
    ThisDisplay.RefreshWindow
    'Rise window from the task bar
    ThisDisplay.MaximizeWindow
```

Sub SetWindowDimensionsPixels

```
Sub SetWindowDimensionsPixels(left As Long, _
     top As Long, _
     width As Long, _
     height As Long)
```

Sets the GraphWorX32 main window size and location in pixels.

Note: Not available for displays running in GWXview32 objects.

VBA example:

```
Sub SetWndDimPixels(o As GwxPick)
    'Declare coordinates as Long values
    Dim viewLeft As Long
    Dim viewTop As Long
    Dim viewWidth As Long
    Dim viewHeight As Long
    'Backup values
    Dim bckLeft As Long
    Dim bckTop As Long
    Dim bckWidth As Long
    Dim bckHeight As Long

    'Remember current dimensions:
```

```
    Call ThisDisplay.GetWindowDimensionsPixels(bckLeft, _
                                               bckTop, _
                                               bckWidth, _
                                               bckHeight)

    'Set new values:
    viewLeft = -2
    viewTop = -2
    viewWidth = 516
    viewHeight = 748

    'Set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPixels(viewLeft, _
                                               viewTop, _
                                               viewWidth, _
                                               viewHeight)
    'and refresh the window:
    ThisDisplay.RefreshWindow

    MsgBox "When you click on OK the view dimensions" + _
        " will be restored to the previous ones." + _
        Chr(13) + " (Left = " + _
        Conversion.CStr(bckLeft) + _
        " , Top = " + _
        Conversion.CStr(bckTop) + _
        " , Width = " + _
        Conversion.CStr(bckWidth) + _
        " and Height = " + _
        Conversion.CStr(bckHeight) + _
        ")"

    'Again, set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPixels(bckLeft, _
                                               bckTop, _
                                               bckWidth, _
                                               bckHeight)
    'and once more refresh the window:
    ThisDisplay.RefreshWindow
End Sub
```

VB example:

```
    'Declare coordinates as Long values
    Dim viewLeft
    Dim viewTop
    Dim viewWidth
    Dim viewHeight
    'Backup values
    Dim bckLeft
    Dim bckTop
    Dim bckWidth
    Dim bckHeight

    'Remember current dimensions:
    Call ThisDisplay.GetWindowDimensionsPixels2(bckLeft, _
                                               bckTop, _
                                               bckWidth, _
                                               bckHeight)
   bckLeft = -4
   bckTop  = -4
   bckWidth = 1160
   bckHeight = 842
```

```
    'New dimensions <0, 0, 502, 341>
    viewLeft = 0
    viewTop = 0
    viewWidth = 502
    viewHeight = 341

    'Set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPixels(viewLeft, _
                                               viewTop, _
                                               viewWidth, _
                                               viewHeight)
    'and refresh the window:
    ThisDisplay.RefreshWindow

    MsgBox "When you click on OK the view dimensions" + _
        " will be restored to the previous ones."

    'Again, set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPixels(bckLeft, _
                                               bckTop, _
                                               bckWidth, _
                                               bckHeight)

    'and once more refresh the window:
    ThisDisplay.RefreshWindow
```

SetWorkingDirectory

```
Sub SetWorkingDirectory()
```

VBA example:

```
    'Sets working directory.

    Dim strWorDir As String
    strWorDir = ThisDisplay.GetWorkingDirectory
    MsgBox "Keep the current directory: " + _
        strWorDir
    ThisDisplay.SetWorkingDirectory (strWorDir)
```

VB example:

```
    'Sets working directory.

    Dim strWorDir
    strWorDir = ThisDisplay.GetWorkingDirectory
    MsgBox "Keep the current directory: " + _
        strWorDir
    ThisDisplay.SetWorkingDirectory (strWorDir)
```

Sub ShowWindow

```
Sub ShowWindow()
```

Shows the GraphWorX32 main window.

VBA example:

```
    'Declare a new GwxDisplay
    Dim newGwxDisplay As New Gwx32.GwxDisplay
    MsgBox "New window is launched..."
```

```
    'Show the display
    Call newGwxDisplay.ShowWindow

    MsgBox "This function works properly only" + _
        " within a dialog which follows."
```

VB example:

```
   'Declare a new GwxDisplay
   Dim newGwxDisplay

   MsgBox "New window is launched..."

   Set newGwxDisplay = CreateObject("Gwx32.Display")

   'Show the display
   Call newGwxDisplay.ShowWindow
```

Sub StartRuntime

```
Sub StartRuntime()
```

Puts GraphWorX32 into runtime mode.

VBA example

```
Sub StartRuntime(o As GwxButton)
    'Switch display to runtime mode.
    ThisDisplay.StartRuntime
End Sub
```

VB example

```
  'You can use it within the function "StopRuntime".

  'Switch to design environment
   ThisDisplay.StopRuntime

   MsgBox "The display is going to quit runtime mode." + _
       " On submit the function StartRuntime is invoked."

   'Switch to runtime environment
   ThisDisplay.StartRuntime
```

Sub StopRuntime

```
Sub StopRuntime()
```

Takes GraphWorX32 out of runtime mode (into configure mode).

VBA example:

```
Sub StopRuntime(o As GwxPick)
    'Switch to design environment
    ThisDisplay.StopRuntime
End Sub
```

VB example:

```
'Switch to design environment
ThisDisplay.StopRuntime
```

Function TestCustomSecurityItem

```
Function TestCustomSecurityItem(customString As String) As Boolean
```

Returns TRUE if the current user has security privilege for the specified custom security item, FALSE otherwise.

GWXSECURITYWRITESTATUS:
WriteUninitialized = -1,
WriteEnabled = 0,
WriteDisabled = 1,
WriteCritical = 2

Function ToggleLayerOverrideHide

```
Function ToggleLayerOverrideHide(layerName As String) As Boolean
```

Toggles the layer override hide property of the specified display layer.

Function ToggleRuntime

```
Function ToggleRuntime() As Boolean
```

Toggles the current operator mode between configure mode and runtime mode. Returns TRUE, if the method put GraphWorX32 into runtime mode, FALSE if the method put GraphWorX32 into configure mode.

VBA example:

```
'Declare flag to watch the mode:
Dim modeFlag As Boolean

'Set the flag according to a current mode:
modeFlag = ThisDisplay.ToggleRuntime
```

VB example:

```
'Declare flag to watch the mode:
Dim modeFlag

'Set the flag according to a current mode:
modeFlag = ThisDisplay.ToggleRuntime
```

Function UngroupSymbol

```
Function UngroupSymbol(objectName As String) As Boolean
```

Ungroups the **GwxSymbol** object with the specified object name. Although the function destroys the specified **GwxSymbol**, it does not destroy the objects that were grouped in that symbol. Returns TRUE for success, FALSE for failure.

Note: Available only in configure mode.

Function VCRControlPanel

```
Function VCRControlPanel(Show As Boolean) As Boolean
```

Opens the VCR control panel.

VBA example:

```
    ThisDisplay.VCRControlPanel (True)
```

Sub ViewBoxZoom

```
Sub ViewBoxZoom()
```

Enters “Box Zoom” mode. This mode allows the user to stretch a box defining the new view area.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
ThisDisplay.ViewBoxZoom
```

VB example:

```
ThisDisplay.ViewBoxZoom
```

Sub ViewFitToWindow

```
Sub ViewFitToWindow()
```

Zooms such that all the objects in the display fit in the view.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    ThisDisplay.ViewFitToWindow

    MsgBox "Zoom back."
    ThisDisplay.ViewHome
```

VB example:

```
    ThisDisplay.ViewFitToWindow

    MsgBox "Zoom back."
    ThisDisplay.ViewHome
```

Sub ViewHome

```
Sub ViewHome()
```

Zooms to the home view. The home view is that view at which the display was last saved.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    Call ThisDisplay.SetViewDimensions(0, 0, 200, 600)
    MsgBox "Zoom back to previous display." + _
        Chr(13) + "(ViewHome)"
    ThisDisplay.ViewHome
```

VB example:

```
    Call ThisDisplay.SetViewDimensions(0, 0, 200, 600)

    MsgBox "Zoom back to previous display." + _
        Chr(13) + "(ViewHome)"
```

```
    ThisDisplay.ViewHome
```

Sub ViewPan

```
Sub ViewPan(offsetX As Long, offsetY As Long)
```

Pans (scrolls) the view by the specified offsets.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    'Set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPercent(0.002, _
                                                0.0025, _
                                                0.45, _
                                                0.5)
    MsgBox "The window pans the view on submit."

    Call ThisDisplay.ViewPan(70, 200)

    MsgBox "Confirm return to the primary display."

    'Restore window dimensions.
    Call ThisDisplay.SetWindowDimensionsPercent(1 / 0.002, _
                                                1 / 0.0025, _
                                                1 / 0.45, _
                                                1 / 0.5)
    ThisDisplay.MaximizeWindow
```

VB example:

```
    'Set the view parameters:
    Call ThisDisplay.SetWindowDimensionsPercent(0.002, _
                                                0.0025, _
                                                0.45, _
                                                0.5)
    MsgBox "The window pans the view on submit."

    Call ThisDisplay.ViewPan(70, 200)

    MsgBox "Confirm return to the primary display."

    'Restore window dimensions.
    Call ThisDisplay.SetWindowDimensionsPercent(1 / 0.002, _
                                                1 / 0.0025, _
                                                1 / 0.45, _
                                                1 / 0.5)
    ThisDisplay.MaximizeWindow
```

Sub ViewShowWholeDisplay

```
Sub ViewShowWholeDisplay()
```

Zooms such that the entire display dimensions fit in the view.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
Sub ViewShowWholeDisplay(o As GwxPick)
    Call ThisDisplay.SetWindowDimensionsPercent(0.5, 0.5, 0.5, 0.5)

    MsgBox "View the whole display."
```

```
    ThisDisplay.ViewShowWholeDisplay

    MsgBox "Restore window."

    ThisDisplay.ViewHome
    ThisDisplay.MaximizeWindow
End Sub
```

VB example:

```
    Call ThisDisplay.SetWindowDimensionsPercent(0.5, 0.5, 0.5, 0.5)

    MsgBox "View the whole display."

    ThisDisplay.ViewShowWholeDisplay

    MsgBox "Restore window."

    ThisDisplay.ViewHome
    ThisDisplay.MaximizeWindow
```

Sub ViewUnzoom

```
Sub ViewUnzoom()
```

Undoes the previous zoom command.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    'The display is 1.5 times bigger.
    ThisDisplay.ViewZoomPercent (0.5)

    MsgBox "Confirm to call ViewUnzoom."

    'Undo zoom.
    ThisDisplay.ViewUnzoom
```

VB example:

```
    'The display is 1.5 times bigger.
    ThisDisplay.ViewZoomPercent (1.5)

    MsgBox "Confirm to call ViewUnzoom."

    'Undo zoom.
    ThisDisplay.ViewUnzoom
```

Sub ViewZoomCustomDialog

```
Sub ViewZoomCustomDialog()
```

Brings up the Custom Zoom dialog. This dialog allows the user to type in a new zoom percentage.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    ThisDisplay.ViewZoomCustomDialog
```

VB example:

```
    ThisDisplay.ViewZoomCustomDialog
```

Sub ViewZoomPercent

```
Sub ViewZoomPercent(newZoom As Single)
```

Zooms in by the specified percentage.

Note: Not available in ActiveX ToolWorX.

VBA example:

```
    Dim Zoom

    Zoom = 2
    Call ThisDisplay.ViewZoomPercent(Zoom)

    MsgBox "Zoom to home."
    ThisDisplay.ViewHome
```

VB example:

```
        Dim Zoom

    Zoom = 2
    Call ThisDisplay.ViewZoomPercent(Zoom)

    MsgBox "Zoom to home."
    ThisDisplay.ViewHome
```

Sub Set VCRNode

```
Sub SetVCRNode(Enable As Boolean)
```

This function allows you to set the location of the VCRWorX server. This is useful, for example, when someone wants to avoid the VCRWorX32 Control Panel and control the VCRWorX32 functionality directly from a display.

Function GetVCRNode

```
Function GetVCRNode() As String
```

This function allows you to get the location of the VCRWorX server. This is useful, for example, when someone wants to avoid the VCRWorX32 Control Panel and control the VCRWorX32 functionality directly from a display.

GwxVisible

GwxVisible is the object type from which visible GraphWorX32 objects (ellipses, rectangles, etc.) are derived. In other words, all visible GraphWorX32 objects have the properties and methods of **GwxVisible.**

Note: See the Create-methods for further explanation of this task.

Properties

Angle

```
Angle As Single
```

Gets/Sets the rotation angle of the object. The angle is specified in degrees.

edgeStyle

```
edgeStyle As GWX3DEDGESTYLE
```

The 3-D edge style of the object's border. Valid values for **GWX3DEDGESTYLE** are:

EdgeNone = 0
EdgeRaised = 5
EdgeEtched = 6
EdgeBump = 9
EdgeSunken = 10

fillColor

```
fillColor As OLE_COLOR
```

Gets/Sets the visible object's fill color.

hasShadow

```
hasShadow As Boolean
```

TRUE if the object has a shadow, FALSE if the object does not have a shadow.

isFilled

```
isFilled As Boolean
```

TRUE if the object is filled, FALSE if the object is not filled.

Keyword

```
Keyword As String
```

This string is used to store a keyword. Use this property to identify an object for GraphWorX32's Update Shared Objects functionality.

lineColor

```
lineColor As OLE_COLOR
```

Gets/Sets the visible object's line/border color.

lineStyle

```
lineStyle As GWXLINESTYLE
```

The style of the object's line/border. Valid values for **GWXLINESTYLE** are:

LineSolid = 0
LineDash = 1
LineDot = 2
LineDashDot = 3

LineDashDotDot = 4
LineNone = 5

lineWidth

```
lineWidth As Long
```

The width of an object's line/border. This value must be in the range of 0 to 10.

objectName

```
objectName As String
```

Gets/Sets the object name of a visible object. The object name is used to identify the object when using certain OLE Automation methods (for instance **GwxDisplay::GetVisibleObjectFromName**). GraphWorX32 will insure that object names are unique. If you assign an object name that already exists for another visible object in a display, GraphWorX32 will append a number to the object name (for example, "tank" would become "tank1", "tank1" would become "tank2", etc.).

Selected

```
Selected As Boolean
```

TRUE if the object is selected, FALSE if the object is not selected. Selected objects are objects which have resize grips around them.

Note: This property can only be changed in configure mode.

shadowColor

```
shadowColor As OLE_COLOR
```

Gets/Sets the visible object's shadow color.

UserCustomData

```
UserCustomData As String
```

This string is used to store custom data. Use this property to associate any additional data with the visible object.

UserDescription

```
UserDescription As String
```

A description string for the visible object. Typically, this string is used to be displayed as informational text in a tooltip.

Visible

```
Visible As Boolean
```

TRUE if the object is visible, FALSE is the object is hidden.

Methods

Sub ClearGradientFill

```
Sub ClearGradientFill()
```

Clears a previously set gradient fill for this visible object.

Sub FlipHorizontal

```
Sub FlipHorizontal()
```

Allows for flipping an object on the horizontal axis. The effect is the same as looking at a horizontal mirror.

Sub FlipVertical

```
Sub FlipVertical()
```

Allows for flipping an object on the vertical axis. The effect is the same as looking at a vertical mirror.

VBA example

```
Dim obj as GWXRectangle
set obj=ThisDisplay.GetVisibleObjectFromName("myobject")
obj.FlipHorizontal
obj.FlipVertical
```

Function GetDataSources

```
Function GetDataSources(includeDuplicates As Boolean)
```

Gets all tag names associated with a visible object. Returns a (variant) array of strings (each string in the array is a tag name). The array can optionally include or not include duplicate tag names.

Function GetDynamicObjectFromIndex

```
Function GetDynamicObjectFromIndex(index As Long) As Object
```

Gets the dynamic object with the specified zero-based index, which is attached to this visible object (index zero is the first attached dynamic). This function is useful for iterating through all the dynamic objects attached to this visible object.

Function GetDynamicObjectFromName

```
Function GetDynamicObjectFromName(nameSubstring As String) As Object
```

Gets the dynamic object with the specified name (or portion of a name), which is attached to this visible object.

Function GetNumberOfDynamics

```
Function GetNumberOfDynamics() As Long
```

Returns the number of dynamic objects attached to this visible object.

Sub GetObjectDimensions

```
Sub GetObjectDimensions(left As Single, top As Single, _
     width As Single, height As Single)
```

Gets the object's size and location. This method does not work in VBScript and Jscript.

Sub GetObjectDimensions2

```
Sub GetObjectDimensions2(left, top, width, height)
```

This method works in Visual Basic for Applications as well as VBScript and JScript. To make it work in Visual Basic for Applications, you need to declare the parameters as Variant.

VB example

```
'VB example, works only in runtime mode
'example for getting object dimensions (GetObjectDimensions2 returns
'variants)
'visible is GwxVisible object
'example requires an object named "test"

Dim l, t, w, h, visible
Set visible = ThisDisplay.GetVisibleObjectFromName("test")
```

```
visible.GetObjectDimensions2 l, t, w, h

MsgBox "Left: " & l & vbCrLf & "Top: " & t & _
vbCrLf & "Width: " & w & _
vbCrLf & "Height: " & h
```

Function GetObjectTypeName

```
Function GetObjectTypeName() As String
```

Returns the type name of this object.

Function GetParentObject

```
Function GetParentObject() As Object
```

This method allows you to get a reference point to the parent object in the GraphWorX32 objects hierarchy. In the following example, imagine you have two objects named "child1" and "child2" grouped together in a symbol named "parent." Starting from the pointer to the child1, you will get a pointer to the parent, and then display is name.

VBA example

```
Dim obj As GwxRectangle
Set obj = ThisDisplay.GetVisibleObjectFromName("child1")
If obj Is Nothing Then
    MsgBox "can't find child1"
Else
    Set parent = obj.GetParentObject
    If parent Is Nothing Then
        MsgBox "can't get parent"
    Else
        MsgBox parent.objectName
    End If
End If
```

Sub MoveObject

```
Sub MoveObject(offsetX As Single, offsetY As Single)
```

Moves the object by the specified offset.

Note: This method only works for objects that are top level visible objects.

Sub RefreshObject

```
Sub RefreshObject()
```

Caused the visible object to be redrawn.

Sub Rotate90

```
Sub Rotate90(clockwise As Boolean)
```

This method rotates objects by 90 degree. It takes one boolean parameter as input to establish if the rotation is clockwise or not. Follow the simple VBA example below on how to rotate a rectangle by 90 degree clockwise.

VBA example

```
Dim rect As GwxRectangle
Set rect = ThisDisplay.GetVisibleObjectFromName("rectangle")
```

```
If rect Is Nothing Then
  MsgBox "An error has occurred while trying to get the object"
Else
  rect.Rotate90 True
End If
```

Function SetAliasDefinition

```
Function SetAliasDefinition(aliasName As String, _
     newDefinition As String) As Boolean
```

Sets the alias definition for all dynamic objects associated with this visible object. If this **GwxVisible** is also a **GwxSymbol**, this function is recursively applied to all objects grouped within the symbol. Returns FALSE if no matching alias names were found, TRUE otherwise.

Note: See also GwxDisplay::SetAliasDefinition and GwxDynamic::SetAliasDefinition.

Function SetAliases

```
Function SetAliases(commandString As String) As Boolean
```

Sets multiple alias definitions for this visible object. If this **GwxVisible** is also a **GwxSymbol**, this function is recursively applied to all objects grouped within the symbol.

Note: See also, GwxDisplay::SetAliases() and GwxVisible::SetAliasDefinition().

Function SetAliasesFromFile

```
Function SetAliasesFromFile(filename As String) As Boolean
```

Sets multiple alias definitions for this visible object. If this **GwxVisible** is also a **GwxSymbol**, this function is recursively applied to all objects grouped within the symbol.

Note: (See also, GwxDisplay::SetAliasesFromFile() and GwxVisible::SetAliasDefinition()).

Function SetGradientFill

```
Function SetGradientFill(style As GWXGRADIENTSTYLE, _
     color1 As OLE_COLOR, _
     color2 As OLE_COLOR, _
     isTwoColor As Boolean, _
     brightness As Single, _
     reverseColors As Boolean, _
     steps As Integer, _
     horizontalOffset As Single, _
     verticalOffset As Single) As Boolean
```

Sets a gradient fill for this visible object.

The parameter **style** can be one of the following values:

GradientHorizontal	= 0
GradientVertical	= 1
GradientSquare	= 2

The rest of the parameters correspond to those in the gradient fill configuration user interface.

Sub SetObjectDimensions

```
Sub SetObjectDimensions(left As Single, _
     top As Single, _
     width As Single, _
     height As Single)
```

Sets the object's size and location.

Note: This method only works for objects that are top level visible objects.

Sub StretchObject

```
Sub StretchObject(scaleX As Single, _
     scaleY As Single, _
     anchorX As Single, _
     anchorY As Single)
```

Stretches the object based on the specified scale factors. The parameters scaleX and scaleY are values such that 0.0 is 0% and 1.0 is 100%. The anchor values are used to change the way the object shifts position during stretching. For example, to resize an object from its center, the anchor values should be the center coordinates of the object.

Note: This method only works for objects that are top level visible objects.

GwxText

GwxText has all the properties and methods of **GwxVisible**, plus the additional properties described below.

Properties

Font

```
Font As IFontDisp
```

Gets/Sets font of the text object.

alignment

```
alignment As GWXTEXTALIGNMENT
```

The alignment of the text (left, center, right). This property is only significant for multiline text strings. Valid values for **GWXTEXTALIGNMENT** are:

TextAlignLeft = 0
TextAlignCenter = 1
TextAlignRight = 2

BoundaryAlignment

```
BoundaryAlignment As GWXTEXTALIGNMENT
```

The alignment of the text relative to the text object's bounding rectangle (left, center, right). Valid values for **GWXTEXTALIGNMENT** are:

TextAlignLeft = 0
TextAlignCenter = 1
TextAlignRight = 2

stretchText

```
stretchText As Boolean
```

TRUE if the font size should be resized when the text object is stretched. FALSE if the font size should remain the same when the object is stretched.

text

```
text As String
```

This string is the text that is displayed by the text object.

GwxRectangle

GwxRectangle has all the properties and methods of **GwxVisible**, plus the additional properties described below.

Properties

rounded

```
rounded As Boolean
```

When getting this property, if the value is TRUE, the rectangle has rounded corners. When setting this property to TRUE, the rectangle is given rounded corners with the default rounding settings.

RoundingX

```
RoundingX As Long
```

Gets/sets the amount of horizontal rounding of the rectangle's corners.

RoundingY

```
RoundingY As Long
```

Gets/sets the amount of vertical rounding of the rectangle's corners.

GwxArc

GwxArc has all the properties and methods of **GwxVisible**, plus the additional properties described below.

Properties

endAngle

```
endAngle As Single
```

Gets/sets the end angle of the arc (in degrees).

arcType

```
arcType As GWXARCTYPE
```

Gets/sets the arc type of the arc. Valid values for **GWXARCTYPE** are:

```
ArcArc      = 0
ArcPie      = 1
ArcChord    = 2
```

startAngle

```
startAngle As Single
```

Gets/sets the start angle of the arc (in degrees).

GwxSymbol

The **GwxSymbol** is used to group together a collection of **GwxVisible** derived objects. **GwxSymbol** objects can be used to group other **GwxSymbol** objects, allowing nested groupings. **GwxSymbol** has all the properties and methods of **GwxVisible**, plus the additional methods described below.

Methods

Function GetDataSourcesRecursive

```
Function GetDataSourcesRecursive(includeDuplicates As Boolean)
```

Gets all tag names associated with a symbol (includes all sub-objects). Returns a (variant) array of strings (each string in the array is a tag name). The array can optionally include or not include duplicate tag names.

Function GetNumberOfChildVisibles

```
Function GetNumberOfChildVisibles() As Long
```

Returns the number of immediate child objects grouped in this symbol.

Function GetVisibleObjectFromIndex

```
Function GetVisibleObjectFromIndex(index As Long) As Object
```

Gets the visible object with the specified zero-based index, which is an immediate child of this symbol (index zero is the first child object). This function is useful for iterating through all the immediate child objects grouped in this symbol.

Function GetVisibleObjectFromName

```
Function GetVisibleObjectFromNaame(nameSubstring As String) As Object
```

Gets the visible object with the specified name (or portion of a name), which is an immediate child of this symbol.

GetVisibleObjectFromPosition

```
GetVisibleObjectFromPosition(x as long, y as long) as Object
```

Returns an object at a specified position in the display. Starts from the root of the display and goes deeper including layers, until the top-most object is found. If the argument **GoIntoSymbol** is set to True, the searching continues until the final visible object at a given position. Otherwise the function returns the top-level symbol that contains the object. The symbol function returns the visible object inside the symbol at the specified position.

VBA example:

```
'VBA example, works in configure and runtime mode
'this example gets the lowest level visible at random position

Dim w As Long, h As Long
Dim o As Object
ThisDisplay.GetDisplayDimensions w, h

'gets display width and height
If ThisDisplay.GetNumberOfTopLevelVisibles > 0 Then

'checks if display has at least one visible object
'and runs a loop for find one
 Do
  Set o = ThisDisplay.GetVisibleObjectFromPosition(Rnd * w, Rnd * h, True)
```

```
'tries to get visible object from random position
'- if object is GwxSymbol, it goes into symbol
'and continues searching
 Loop Until Not (o Is Nothing)
'after any object is found, loop ends and example
'shows name of the object
 MsgBox o.objectName
Else
 MsgBox "Example needs at least one visible object"
End If
```

VB example:

```
    Dim obj

    Set obj = ThisDisplay.GetVisibleObjectFromPosition(588, _
                                                       256, _
                                                       True)
    Dim rect
    Set rect = obj
    MsgBox "Change the color of the rectangle above." + _
        " Its name is: " + CStr(rect.objectName)

    rect.fillColor = RGB(200, 30, 255)

    MsgBox "Undo fill color."

    rect.fillColor = RGB(200, 200, 255)
```

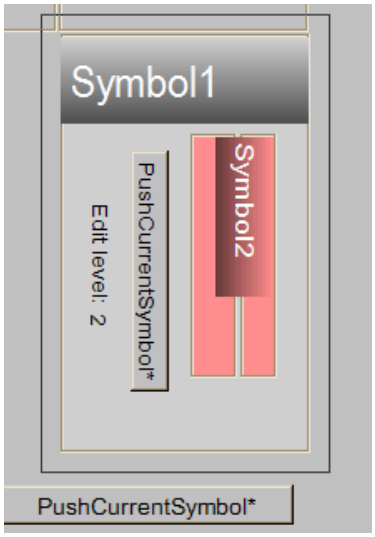
Function PushCurrentSymbol

```
Function PushCurrentSymbol() As Boolean
```

Pushes this symbol object onto the symbol edit stack. This essentially makes the objects grouped in this symbol, top level visible objects

Note: Some operations, like **GwxVisible::SetObjectDimensions** are only allowed on top level objects.

VBA example:




```
Dim obj As Object

MsgBox "PushCurrentSymbol, ""Symbol1""" + _
    Chr(13) + "Press the button ""PushCurrentSymbol2""" + _
    " to reach the Symbol2 in the Symbol1."

Set obj = ThisDisplay.GetVisibleObjectFromName("Object1")
Dim oRec As GwxSymbol
Set oRec = obj
oRec.PushCurrentSymbol

Set obj = ThisDisplay.GetVisibleObjectFromName("EditLevel")
obj.text = CStr(Val(obj.text) + 1)

MsgBox "See the current edit level." + _
        Chr(13) + _
        "We can double-click aside from this symbol or " + _
        "we can call the function PopCurrentSymbol()."
```

GwxOleObject

GwxOleObject has all the properties and methods of **GwxVisible**, plus the additional methods described below.

Note: GwxOleObjects used with ActiveX ToolWorX cannot be activated and consequently should not be used.

Methods

Function GetOLEObject

```
Function GetOLEObject() As Object
```

Gets the actual OLE object wrapped by **GwxOleObject.** The method allows access to the embedded ActiveX object. For example, assume that a TrendWorX32 Viewer object called "twx_01" (the name was assigned in the GraphWorX32 Property Inspector) is inserted into the GraphWorX32 display. Then it is possible to use the following code to access this ActiveX control in runtime and work with it:

VBA example:

```
Dim o As GwxOleObject
Set o = ThisDisplay.GetVisibleObjectFromName("twx_01")

Dim twx As TWXVIEWERLib.TWXViewer
Set twx = o.GetOLEObject()

twx.BackColor = RGB(0, 255, 0)
```

GwxEllipse

GwxEllipse has all the properties and methods of **GwxVisible**.

GwxLine

GwxLine has all the properties and methods of **GwxVisible**.

GwxBitmap

GwxBitmap has all the properties and methods of **GwxVisible**.

GwxMetafile

GwxMetafile has all the properties and methods of **GwxVisible**.

GwxButton

A **GwxButton** object is not fully functional until it has at least one **GwxPick** dynamic object associated with it. **GwxButton** has all the properties and methods of **GwxVisible** and **GwxText**.

GwxDynamic

GwxDynamic is the object type from which dynamic GraphWorX32 objects (size dynamic, location dynamic, etc.) are derived. In other words, all dynamic GraphWorX32 objects have the properties and methods of **GwxDynamic**.

Properties

dataSource

```
dataSource As String
```

The primary data source for the dynamic object. **DataSource** is a string which represents an OPC tag name, an expression, a constant value, or a GraphWorX32 local variable.

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

Dim strName As String
strName = oPP.GetObjectTypeName

oPP.UserDescription = "ExampleDataSource"

MsgBox "Changing data source in the ProcessPoint to gfwsim.cosine.double."
oPP.dataSource = "gfwsim.cosine.double"
```

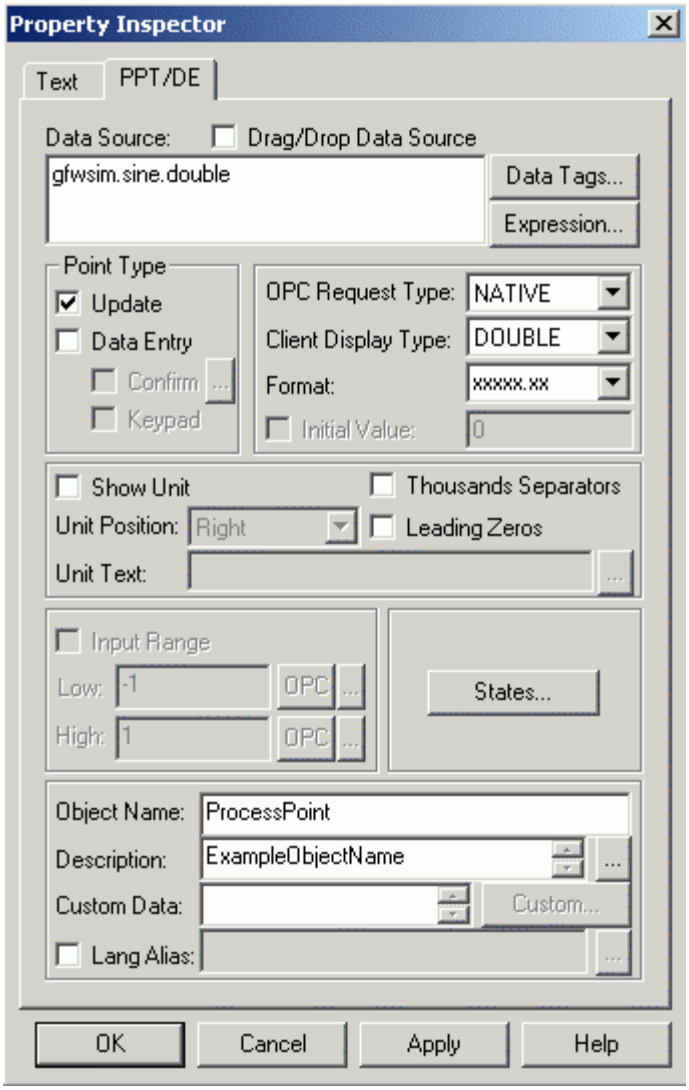
VB example:

```
Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim strName
strName = obj.GetObjectTypeName

obj.UserDescription = "ExampleDataSource"

MsgBox "Changing data source in the ProcessPoint to gfwsim.cosine.double."
obj.dataSource = "gfwsim.cosine.double"
```



highRange

```
highRange As String
```

Represents the overridden high range for this dynamic. **HighRange** is a string which represents an OPC tag name, an expression, a constant value, or a GraphWorX32 local variable. This property is only used for dynamics based on an analog data source; it is ignored for dynamics which are based on digital connections.

VBA example:

```
Dim obj As Object
 Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

 Dim oDynamic As GwxDynamic
 'GwxProcessPoint has all the properties
 'and methods of GwxDynamic.

 Set oDynamic = obj
 'We can complete code via intellisence...

 Dim oPP As GwxProcessPoint
```

```
Set oPP = oDynamic

Dim strName As String
strName = oPP.GetObjectTypeName

oPP.UserDescription = "ExampleHighRange"
Dim h As String
Dim l As String
h = oPP.highRange
l = oPP.lowRange

'This properties are ignored for
'dynamics which are based on an analog data
'source.
MsgBox "Set ranges to <-0.5;0.5> units."


oPP.RangeOverride = True

oPP.highRange = "0.5"
oPP.lowRange = "-0.5"
```

VB example:

```
MsgBox "This method is valid only for Data Entry."

Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim oDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP
Set oPP = oDynamic

Dim strName
strName = oPP.GetObjectTypeName

oPP.UserDescription = "ExampleHighRange"
Dim h
Dim l
h = oPP.highRange
l = oPP.lowRange

'This properties are ignored for
'dynamics which are based on an analog data
'source.
MsgBox "Set ranges to <-0.5;0.5> units."

oPP.RangeOverride = True

oPP.highRange = "0.5"
oPP.lowRange = "-0.5"
```

lowRange

```
lowRange As String
```

Represents the overridden low range for this dynamic. **LowRange** is a string which represents an OPC tag name, an expression, a constant value, or a GraphWorX32 local variable. This property is only used for dynamics based on an analog data source; it is ignored for dynamics which are based on digital connections.

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

Dim strName As String
strName = oPP.GetObjectTypeName

oPP.UserDescription = "ExampleLowRange"
Dim h As String
Dim l As String
h = oPP.highRange
l = oPP.lowRange

'This properties are ignored for
'dynamics which are based on an analog data
'source.

MsgBox "Set ranges to <-0.5;0.5> units."

oPP.highRange = "0.5"
oPP.lowRange = "-0.5"
```

VB example:

```
MsgBox "This method is valid only for Data Entry."

Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim oDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP
Set oPP = oDynamic

Dim strName
strName = oPP.GetObjectTypeName

oPP.UserDescription = "ExampleLowRange"
Dim h
Dim l
h = oPP.highRange
```

```
l = oPP.lowRange

'This properties are ignored for
'dynamics which are based on an analog data
'source.

MsgBox "Set ranges to <-0.5;0.5> units."

oPP.highRange = "0.5"
oPP.lowRange = "-0.5"
```

objectName

```
objectName As String
```

Gets/Sets the object name of a dynamic object. The object name is used to identify the object when using certain OLE Automation methods (for instance **GwxDisplay::GetDynamicObjectFromName**). GraphWorX32 will insure that object names are unique. If you assign an object name that already exists for another dynamic object in a display, GraphWorX32 will append a number to the object name (for example, “size” would become “size1”, “size1” would become “size2”, etc.).

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

Dim strName As String
strName = oPP.GetObjectTypeName

oPP.UserDescription = "ExampleObjektName"
oPP.objectName = "ProcessPoint"
```

VB example:

```
Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")


Dim strName
strName = obj.GetObjectTypeName

obj.UserDescription = "ExampleObjektName"
MsgBox "The name of the process point above is: " + _
   obj.objectName
```

RangeOverride

```
RangeOverride As Boolean
```

Gets/Sets range override status. When FALSE, GraphWorX32 will use the ranges associated with the primary **DataSource**. When TRUE, GraphWorX32 will use the ranges defined in the **HighRange** and **LowRange** properties. This property is only used for dynamics based on an

analog data source (**GwxSize**, **GwxLocation**, **GwxRotation**, **GwxAnalogColor**, **GwxAnalogSelector**, **GwxProcessPoint**); it is ignored for dynamics which are based on digital connections.

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

Dim strName As String
strName = oPP.GetObjectTypeName

oPP.UserDescription = "ExampleRangeOverride"

oPP.RangeOverride = True
oPP.highRange = "0.5"
oPP.lowRange = "-0.5"
```

VB example:

```
Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

obj.RangeOverride = True
oPP.highRange = "0.5"
oPP.lowRange = "-0.5"
```

RequestDataType

```
RequestDataType As GWXDATATYPE
```

Allows you to specify the datatype used to request the data source from the OPC server.

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

Dim strName As String
strName = oPP.GetObjectTypeName
```

```
oPP.UserDescription = "ExampleRequestDataType"

oPP.RequestDataType = DataTypeDouble
```

VB example:

```
Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

obj.UserDescription = "ExampleRequestDataType"

obj.RequestDataType = DataTypeDouble
```

TimerRate

```
TimerRate As Long
```

Frequency update rate for timer based dynamic types (this property is ignored for dynamics which are not timer based). Timer based dynamics include: **GwxFlash**, **GwxAnimator**, and **GwxPick**.

VBA example:

```
Dim obj As Object
 Set obj = ThisDisplay.GetDynamicObjectFromName("AnimatedCar")

 Dim oDynamic As GwxDynamic
 'GwxProcessPoint has all the properties
 'and methods of GwxDynamic.

 Set oDynamic = obj
 'We can complete code via intellisence...

 Dim oAni As GwxAnimator
 Set oAni = oDynamic

 Dim strName As String
 strName = oAni.GetObjectTypeName

 oAni.TimerRate = 1000 'ms
 MsgBox "See the animated car with the frame rate " + _
     CStr(oAni.TimerRate) + " ms."

 oAni.TimerRate = 50

 MsgBox "Now the frame rate has changed to 50 ms."
```

VB example:

```
Dim obj
 Set obj = ThisDisplay.GetDynamicObjectFromName("AnimatedCar")


 Dim strName
 strName = obj.GetObjectTypeName

 obj.TimerRate = 1000 'ms
 MsgBox "See the animated car with the frame rate " + _
     CStr(obj.TimerRate) + " ms."

 obj.TimerRate = 50
```

```
    MsgBox "Now the frame rate has changed to 50 ms."
```

UserCustomData

```
UserCustomData As String
```

This string is used to store custom data. Use this property to associate any additional data with the dynamic object.

VB example:

```
   Dim obj
    Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")


    obj.UserDescription = "ExampleCustomdata"
    obj.UserCustomData = "Device #XY"
```

VBA example:

```
    Dim obj As Object
    Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

    Dim oDynamic As GwxDynamic
    'GwxProcessPoint has all the properties
    'and methods of GwxDynamic.

    Set oDynamic = obj
    'We can complete code via intellisence...

    Dim oPP As GwxProcessPoint
    Set oPP = oDynamic

    Dim strName As String
    strName = oPP.GetObjectTypeName

    oPP.UserDescription = "ExampleCustomdata"
    oPP.UserCustomData = "Device #XY"
```

UserDescription

```
UserDescription As String
```

A description string for the dynamic object. Typically, this string is used to be displayed as informational text in a tooltip.

VBA example:

```
    Dim obj As Object
    Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

    Dim oDynamic As GwxDynamic
    'GwxProcessPoint has all the properties
    'and methods of GwxDynamic.

    Set oDynamic = obj
    'We can complete code via intellisence...

    Dim oPP As GwxProcessPoint
    Set oPP = oDynamic

    Dim strName As String
```

```
    strName = oPP.GetObjectTypeName

    oPP.UserDescription = "ExampleUserDescription"
```

Methods

Function GetAliasDefinition

```
Function GetAliasDefinition(aliasName As String) As String
```

Gets the alias definition of the specified alias name.

VBA example:

```
    Dim obj As Object
    Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPointWithAlias")

    Dim oDynamic As GwxDynamic
    'GwxProcessPoint has all the properties
    'and methods of GwxDynamic.

    Set oDynamic = obj
    'We can complete code via intellisence...

    Dim oPP As GwxProcessPoint
    Set oPP = oDynamic
    Dim p
    p = oPP.objectName
    Dim strName As String
    strName = oPP.GetObjectTypeName

    Dim strNewAliasDefinition As String
    Dim strAliasName As String
    strAliasName = "AliasString"


    'Call oPP.SetAliasDefinition(strAliasName, "gfwsim.cosine.double")

    strNewAliasDefinition = oPP.GetAliasDefinition(strAliasName)
    MsgBox "Alias definition of the ""ProcessPoint"" is: " + _
        strNewAliasDefinition
```

VB example:

```
    Dim obj
    Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPointWithAlias")

    Dim strName
    strName = obj.GetObjectTypeName

    Dim strNewAliasDefinition
    Dim strAliasName
    strAliasName = "AliasString"


    'Call oPP.SetAliasDefinition(strAliasName, "gfwsim.cosine.double")

    strNewAliasDefinition = obj.GetAliasDefinition(strAliasName)
    MsgBox "Alias definition of the ""ProcessPoint"" is: " + _
        strNewAliasDefinition
```

Function GetDataSourcePointObject

```
Function GetDataSourcePointObject() As Object
```

Gets the **GwxPoint** object for the data source of this dynamic object.

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

Dim strName As String
strName = oPP.GetObjectTypeName

Dim oPoint As GwxPoint
Set oPoint = oPP.GetDataSourcePointObject

Dim str As String

MsgBox "The Process Point includes an object - GwxPoint." + _
    Chr(13) + "The data type number of the point is: " + _
    CStr(oPoint.dataType)
```

VB example:

```
Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim oDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP
Set oPP = oDynamic

Dim strName
strName = oPP.GetObjectTypeName

Dim oPoint
Set oPoint = oPP.GetDataSourcePointObject

Dim str

MsgBox "The Process Point includes an object - GwxPoint." + _
    Chr(13) + "The data type number of the point is: " + _
    CStr(oPoint.dataType)
```

Function GetHighRangePointObject

```
Function GetHighRangePointObject() As Object
```

Gets the **GwxPoint** object for the high range of this dynamic object.

VBA example:

```
    MsgBox "It has no affect untill you use property" + _
           "highRange"
   'Suppose we have created a process point with
   'the data entry checked off. (e.g. <-100; 100>)
   'We will change values of ranges to <-0.5; 0.5>.


   Dim obj As Object
   Set obj = _
     ThisDisplay.GetDynamicObjectFromName("DataEntryProcessPoint")

   Dim oDynamic As GwxDynamic
   'GwxProcessPoint has all the properties
   'and methods of GwxDynamic.

   Set oDynamic = obj
   'We can complete code via intellisence...

   Dim oPP As GwxProcessPoint
   Set oPP = oDynamic

   Dim oHRPoint As GwxPoint

   Set oHRPoint = oPP.GetHighRangePointObject
   Dim strValue As String

   strValue = oHRPoint.GetPointName
   MsgBox "The High Range Point Object of” + _
    “ the ""Process Point"" is: " + strValue
```

VB example:

```
   'Suppose we have created a process point with
   'the data entry checked off. (e.g. <-100; 100>)
   'We will change values of ranges to <-0.5; 0.5>.

   MsgBox "It has no affect untill you use property" + _
      "highRange"

   Dim obj
   Set obj = _
       ThisDisplay.GetDynamicObjectFromName("DataEntryProcessPoint")

   Dim oHRPoint
   Set oHRPoint = obj.GetHighRangePointObject

 Dim strValue
   strValue = oHRPoint.GetPointName
   MsgBox "The High Range Point Object of the ""Process Point"" is: " + _
        strValue
```

Function GetLowRangePointObject

```
Function GetLowRangePointObject() As Object
```

Gets the **GwxPoint** object for the low range of this dynamic object.

VBA example:

```
'Suppose we have created a process point with
'the data entry checked off. (e.g. <-100; 100>)


Dim obj As Object

Set obj = _
  ThisDisplay.GetDynamicObjectFromName("DataEntryProcessPoint")

Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

Dim oHLPoint As GwxPoint

Set oHLPoint = oPP.GetLowRangePointObject

Dim strValue As String
strValue = oHLPoint.GetPointName

MsgBox "The Low Range Point Object of the ""Process Point"" is: " + _
    strValue
```

VB example:

```
MsgBox "It has no affect untill you use property" + _
   "lowRange"
'Suppose we have created a process point with
'the data entry checked off. (e.g. <-100; 100>)

Dim obj
Set obj = _
    ThisDisplay.GetDynamicObjectFromName("DataEntryProcessPoint")

 Dim  oHLPoint
Set oHLPoint = obj.GetLowRangePointObject

Dim strValue
strValue = oHLPoint.GetPointName

MsgBox "The Low Range Point Object of the ""Process Point"" is: " + _
    strValue
```

Function GetNumberOfSubDynamics

```
Function GetNumberOfSubDynamics() As Long
```

Returns the number of immediate child sub-dynamic objects used in this dynamic object. Sub-dynamic objects are used by the dynamic types: **GwxDigitalColor**, **GwxDigitalSelector**, and **GwxPick**.

VBA example:

```
'We sssume we have added a color dynamic
'to a rectangle with several tags.
```

```
    'gfwsim.sine.double, etc.

    Dim obj As Object
    Set obj = ThisDisplay.GetDynamicObjectFromName("SubDynamic")

    Dim oDynamic As GwxDynamic
    'GwxProcessPoint has all the properties
    'and methods of GwxDynamic.

    Set oDynamic = obj
    'We can complete code via intellisence...

    Dim oPick As GwxDigitalColor
    Set oPick = oDynamic

    Dim lNumberOfSubDynamics As Long
    lNumberOfSubDynamics = oPick.GetNumberOfSubDynamics

    MsgBox "The number of immediate child sub-dynamics in" + _
           " GwxDigitalColor rectangle is: " + _
        CStr(lNumberOfSubDynamics)
```

VB example:

```
    'We sssume we have added a color dynamic
    'to a rectangle with several tags.
    'gfwsim.sine.double, etc.

    Dim obj
    Set obj = ThisDisplay.GetDynamicObjectFromName("SubDynamic")

    Dim lNumberOfSubDynamics
    lNumberOfSubDynamics = obj.GetNumberOfSubDynamics

    MsgBox "The number of immediate child sub-dynamics in" + _
           " GwxDigitalColor rectangle is: " + _
        CStr(lNumberOfSubDynamics)
```

Function GetObjectTypeName

```
Function GetObjectTypeName() As String
```

Returns the type name of this object.

VBA example:

```
    Dim obj As Object
    Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

    Dim oDynamic As GwxDynamic
    'GwxProcessPoint has all the properties
    'and methods of GwxDynamic.

    Set oDynamic = obj
    'We can complete code via intellisence...

    Dim oPP As GwxProcessPoint
    Set oPP = oDynamic

    MsgBox "The type name of the Process Point is: " + _
        oPP.GetObjectTypeName
```

VB example:

```
Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")


MsgBox "The type name of the Process Point is: " + _
    obj.GetObjectTypeName
```

Function GetSubDynamicObjectFromIndex

```
Function GetSubDynamicObjectFromIndex(index As Long) As Object
```

Gets the sub-dynamic object with the specified zero-based index, which is an immediate child of this dynamic object (index zero is the first child object).

Note: See also GetNumberOfSubDynamics().

VBA example:

```
'We sssume we have added a color dynamic
'to a rectangle with several tags.
'gfwsim.sine.double, etc.

Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("SubDynamic")

Dim oDynamic As GwxDynamic
'GwxDigitalColor has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oDC As GwxDigitalColor
Set oDC = oDynamic

Dim obj2
Set obj2 = oDC.GetSubDynamicObjectFromIndex(2)
```

VB example:

```
'We sssume we have added a color dynamic
'to a rectangle with several tags.
'gfwsim.sine.double, etc.

Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("SubDynamic")


Dim obj2
Set obj2 = obj.GetSubDynamicObjectFromIndex(2)
```

Function GetVisibleObject

```
Function GetVisibleObject() As Object
```

Gets the visible object to which this dynamic is attached.

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")
'We assume that the process point has the name "ProcessPoint"
```

```
Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

Dim obj2 As Object
Set obj2 = oPP.GetVisibleObject

MsgBox "The name of the visible object is: " + _
    obj2.objectName
```

VB example:

```
Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")
'We assume that the process point has the name "ProcessPoint"

Dim obj2
Set obj2 = obj.GetVisibleObject

MsgBox "The name of the visible object is: " + _
    obj2.objectName
```

Function SetAliasDefinition

```
Function SetAliasDefinition(aliasName As String, _
     newDefinition As String) As Boolean
```

Sets the alias definition for this dynamic object only. Returns FALSE if no matching alias names were found, TRUE otherwise.

Note: See also GwxDisplay::SetAliasDefinition and wxVisible::SetAliasDefinition.

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPointWithAlias")

Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

Dim strAliasName As String
strAliasName = "AliasString"

Call oPP.SetAliasDefinition(strAliasName, "{{gfwsim.cosine.double}}*20")
MsgBox "The new definition for the alias" + _
    "in the Process Point is {{gfwsim.cosine.double}}*20."
```

VB example:

```
Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPointWithAlias")


Dim strAliasName
strAliasName = "AliasString"

Call obj.SetAliasDefinition(strAliasName, "{{gfwsim.cosine.double}}*20")
MsgBox "The new definition for the alias" + _
    "in the Process Point is {{gfwsim.cosine.double}}*20."
```

Function SetAliases

```
Function SetAliases(commandString As String) As Boolean
```

Sets multiple alias definitions for this dynamic object.

Note: See also, GwxDisplay::SetAliases() and GwxDynamic::SetAliasDefinition().

VBA example:

```
Dim obj As Object
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPointWithAlias")

Dim oDynamic As GwxDynamic
'GwxProcessPoint has all the properties
'and methods of GwxDynamic.

Set oDynamic = obj
'We can complete code via intellisence...

Dim oPP As GwxProcessPoint
Set oPP = oDynamic

'Compose a command string:
'See GwxDisplay::SetAliasDefinition
Dim strAliasCommand As String
strAliasCommand = "AliasString" + Chr(9) + _
                    "100+abs({{gfwsim.sine.double}})*500"

MsgBox "Now we define another source for the GwxProcessPoint."
Call oPP.SetAliases(strAliasCommand)
```

VB example:

```
Dim obj
Set obj = _
ThisDisplay.GetDynamicObjectFromName("ProcessPointWithAlias")


'Compose a command string:
'See GwxDisplay::SetAliasDefinition
Dim strAliasCommand
strAliasCommand = "AliasString" + Chr(9) + _
                    "100+abs({{gfwsim.sine.double}})*500"

MsgBox "Now we define another source for" + _
   " the GwxProcessPoint." + _
  "100+Abs({{gfwsim.sine.double}})*500"

Call obj.SetAliases(strAliasCommand)
```

Function SetAliasesFromFile

```
Function SetAliasesFromFile(filename As String) As Boolean
```

Sets multiple alias definitions for this dynamic object.

Note: See also, GwxDisplay::SetAliasesFromFile() and wxDynamic::SetAliasDefinition().

VBA example:

```
Dim strTextFileName As String

 'Specify a text file. Take care all
 'of the white spaces (only TABS) dividing the strings.
 'Set to use real tabs in your editor.
 'Template is:
 'aliasName1<TAB>aliasDef1<CR-LF>
 'aliasName2<TAB>aliasDef2<CR-LF>
 '....
 '..
 '.
 strTextFileName = "AliasCommandString.txt"

 MsgBox "Now we define another source for the GwxProcessPoint."


 Dim obj As Object
 Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPointWithAlias")

 Dim oDynamic As GwxDynamic
 'GwxProcessPoint has all the properties
 'and methods of GwxDynamic.

 Set oDynamic = obj
 'We can complete code via intellisence...

 Dim oPP As GwxProcessPoint
 Set oPP = oDynamic


 Call oPP.SetAliasesFromFile(strTextFileName)
```

VB example:

```
 Dim strTextFileName

 'Specify a text file. Take care all
 'of the white spaces (only TABS) dividing the strings.
 'Set to use real tabs in your editor.
 'Template is:
 'aliasName1<TAB>aliasDef1<CR-LF>
 'aliasName2<TAB>aliasDef2<CR-LF>
 '....
 '..
 '.
 strTextFileName = "AliasCommandString.txt"

 MsgBox "Now we define another source for the GwxProcessPoint."


 Dim obj
 Set obj = _
```

```
      ThisDisplay.GetDynamicObjectFromName("ProcessPointWithAlias")


   Call obj.SetAliasesFromFile(strTextFileName)
```

GwxDigitalSelector

The **GwxDigitalSelector** object is used to hide/show individual objects from a set of objects based on a digital (boolean) signal. **GwxDigitalSelector** objects are only associated with **GwxSymbol** objects (the **GwxSymbol** object defines the set of objects upon which the **GwxDigitalSelector** object acts). **GwxDigitalSelector** has all the properties and methods of **GwxDynamic**, plus the additional method described below.

Methods

Function SetConnectionInfo

```
Function SetConnectionInfo(objectNumber As Integer, _
     dataSource As String, _
     showWhenTrue As Boolean) As Boolean
```

Sets data connections for this digital selector object. The parameter **objectNumber** is a zero-based index representing an immediate child of this symbol object to which this dynamic is attached (index zero is the first child object).

Note: See also GwxDigitalSelectorInfo.

GwxDigitalSelectorInfo

This object includes properties of individual data connections for **GwxDigitalSelector** objects. **GwxDigitalSelectorInfo** has all the properties and methods of **GwxDynamic**.

GwxDigitalColor

The **GwxDigitalColor** object is used to change the color of the associated visible object based on one or more digital (boolean) signals. **GwxDigitalColor** has all the properties and methods of **GwxDynamic**, plus the additional methods described below.

Methods

Function DeleteSubDynamic

```
Function DeleteSubDynamic(index As Integer) As Boolean
```

This function deletes subdynamic object at the position index. It returns true on success. It returns false on failure, typically when the index is out of range.

Function SwapSubDynamics

```
Function SwapSubDynamics(index1 As Integer, _
     index2 As Integer) As Boolean
```

This function swaps subdynamic objects at the positions index1 and index2. It returns true on success. It returns false on failure, typically when the index is out of range.

GwxDigitalColorInfo

The **GwxDigitalColorInfo** object includes properties of individual data connections for **GwxDigitalColor** objects. **GwxDigitalColorInfo** has all the properties and methods of **GwxDynamic,** plus the additional properties described below.

Properties

UseFillColor

```
UseFillColor As Boolean
```

This property enables color changes on the visual object:

UseLineColor

```
UseLineColor As Boolean
```

This property enables color changes on the visual object:

UseShadowColor

```
UseShadowColor As Boolean
```

This property enables color changes on the visual object:

FillColor

```
FillColor As OLE_COLOR
```

This property defines the colors for the color action:

LineColor

```
LineColor As OLE_COLOR
```

This property defines the colors for the color action:

ShadowColor

```
ShadowColor As OLE_COLOR
```

This property defines the colors for the color action:

ChangeColorOnTrue

```
ChangeColorOnTrue As Boolean
```

This property specifies if the action is fired when the OPC tag is evaluated to true or false.

GwxAnalogSelector

The **GwxAnalogSelector** object is used to hide/show individual objects from a set of objects based on an analog signal. **GwxAnalogSelector** objects are only associated with **GwxSymbol** objects (the **GwxSymbol** object defines the set of objects upon which the **GwxAnalogSelector** object acts). **GwxAnalogSelector** has all the properties and methods of **GwxDynamic**, plus the additional method described below.

Methods

Function SetSelectorRanges

```
Function SetSelectorRanges(rangeArray) As Boolean
```

Sets selector ranges for this dynamic. Returns TRUE for success, FALSE for failure.

The parameter **rangeArray** is a SAFEARRAY of floats. The number of elements in the array must equal the number of objects in the analog selector. Each index in the array corresponds to the index of an object in the analog selector. Each array element represents a percentage value for the high limit of that object's range. Each array element must be a value in the range 0 to 1. Each array element value must be greater than the previous element value. The last array element's value must always be 1.

GwxAnalogColor

The **GwxAnalogColor** object is used to change the color of the associated visible object based on an analog signal. **GwxAnalogColor** has all the properties and methods of **GwxDynamic**.

GwxAnimator

The **GwxAnimator** object is used to hide/show individual objects from a set of objects in a specified sequence, based on elapsed time. The **GwxAnimator** is activated/deactivated based on the value of a digital signal. **GwxAnimator** objects are only associated with **GwxSymbol** objects (the **GwxSymbol** object defines the set of objects upon which the **GwxAnimator** object acts). **GwxAnimator** has all the properties and methods of **GwxDynamic**.

GwxSize

The **GwxSize** object is used to change the size of the associated visible object based on the value of an analog signal. **GwxSize** has all the properties and methods of **GwxDynamic**.

GwxLocation

The **GwxLocation** object is used to change the location of the associated visible object based on the value of an analog signal. **GwxLocation** has all the properties and methods of **GwxDynamic,** plus the additional methods described below.

Methods

Sub GetCoordinateFromLocationDelta

```
Sub GetCoordinateFromLocationDelta(offset As Single, cx, cy)
```

This is an advanced functionality added for the advanced user who wants to develop complex applications. General applications will not find a use for this function.

This function allows you to retrieve the coordinate that an object would have on a Location dynamic if the tag associated with the dynamic had a specific value (offset).

This method is very handy to determine the position that an object would have on a complex path such as a train track, starting from a known offset. It basically performs a transformation from a linear space (offset on the path) to a bidimensional space (coordinate on the display).

This method uses the same algorithm used for the location dynamic to determine the position of an object on the location path based on the tag value.

Using this method in combination with “setobjectdimension” it is possible to change the position of several objects in a display and relocate them on a track.

The minimum offset and maximum offset values are the same as the min and max range of the location dynamic.

Function GetLastEnteredValue

```
Function GetLastEnteredValue()
```

Returns the last entered value, which might be different from the currently displayed value.

GwxRotation

The **GwxRotation** object is used to change the angle of the associated visible object based on the value of an analog signal. **GwxRotation** has all the properties and methods of **GwxDynamic,** plus the additional methods described below.

Methods

Function GetLastEnteredValue

```
Function GetLastEnteredValue()
```

Returns the last entered value, which might be different from the currently displayed value.

GwxFlash

The **GwxFlash** object is used to toggle the color or visibility of the associated visible object at a specified rate. The **GwxFlash** object is activated/deactivated based on the value of a digital signal. **GwxFlash** has all the properties and methods of **GwxDynamic**.

GwxHide

The **GwxHide** object is used to toggle the visibility or enabled/disabled state of the associated visible object based on the value of a digital signal. The **GwxHide** has all the properties and methods of **GwxDynamic**.

GwxPick

Associating a **GwxPick** object with a visible object designates that the associated visible object will perform an active when clicked on by the operator during runtime mode. **GwxPick** objects can be associated with **GwxButton** objects to make fully functional pushbuttons. **GwxPick** has all the properties and methods of **GwxDynamic**.

Properties

aliasCommandString

```
aliasCommandString As String
```

If the **AliasCommandType** property is **AliasSetFromFile**, **AliasCommandString** should be a filename that specifies a text file containing tab-separated pairs of alias names and alias definitions.

Note: see also SetAliasesFromFile().

If **AliasCommandType** is **AliasSetDirect**, **AliasCommandString** should have the following format:

aliasName1<TAB>aliasDef1<CR-LF>aliasName2<TAB>aliasDef2<CR-LF> … etc.

aliasCommandType

```
aliasCommandType As GWXALIASCOMMAND
```

AliasCommandType can be one of the following values:

AliasNoCommand = 0
AliasSetFromFile = 1
AliasSetDirect = 2

Note: See also *aliasCommandString* for further details.

NameParameter

```
NameParameter As String
```

The meaning of name parameter varies based on the value of the **PickAction** property.

If **PickAction** equals **PickLoadDisplay**, **PickDragDropLoad**, **PickPopupWindow**, or **PickEmbeddedWindow**, **NameParamter** is the Gwx32 display filename (.gdf).

If **PickAction** equals **PickLaunchApp**, **NameParameter** is an executable filename (.exe).

If **PickAction** equals **PickRunScript**, **NameParamter** is a VBA macro name.

If **PickAction** equals **PickSetAliases** or **PickAliasesDialog**, **NameParameter** is a visible object name.

pickAction

```
pickAction As GWXPICKACTION
```

Note: See **GwxDisplay::CreatePickDynamic** for description of this property.

ScriptText

```
ScriptText As String
```

GraphWorX32 allows for associating VBScript and JScript with a pick action. The scripts associated with the pick action are executed when the pick event is triggered (for example, when the left mouse button is pressed). These scripts can be edited through the new integtated script toolbar or through the OLE automation interface. Each script can be written using VBScript or JScript language independently from the other scripts associated with the other pick action. This property allows for reading and writing the script source code currently associated with the pick action.

ScriptType

```
ScriptType As GWXSCRIPTTYPE
```

GraphWorX32 allows for associating VBScript and JScript with a pick action. The scripts associated with the pick action are executed when the pick event is triggered (for example, when the left mouse button is pressed). These scripts can be edited through the new integtated script toolbar or through the OLE automation interface. Each script can be written using VBScript or JScript language independently from the other scripts associated with the other pick action. This property allows for getting and settind the script language currently associated with the pick action.

VBA example:

```
'VBA example, works only in configure mode
'example for creating pick action PickRunScript and set script text
'and type

Dim o As Object

'creates rectangle
Set o = ThisDisplay.CreateRectangle( _
50, 50, 100, 100, True, 100, 0, 2, _
LineSolid, False, 0, EdgeNone, False, _
"rectangle", False)

'adds pick action PickRunScript on the rectangle
Set o = ThisDisplay.CreatePickDynamic( _
"rectangle", "dynamic", PickRunScript, _
ButtonNormal, TriggerOnDown, MouseButtonLeft, _
False, "", "", False, False, "", "", "")

'sets script properties - name of the script is
'derived from pick action name
o.ScriptType = ScriptTypeVBScript
o.ScriptText = "MsgBox " & Chr(34) & "Test OK" & Chr(34)
```

tracking

```
tracking As Boolean
```

If the pick dynamic is a toggle or download pick and a checkbox or radio button pick, setting tracking to TRUE will cause the button state to track the data value associated with this pick dynamic.

Methods

Sub SimulateClick

```
Sub SimulateClick(mouseButton As GWXMOUSEBUTTON)
```

Causes this pick dynamic to behave as though the user clicked on this pick with the specified mouse button.

Note: You should validate the result of this function before running additional code, and should avoid sending bursts of display load requests. A new display load request might arrive before the current display load request is completed.

GwxProcessPoint

GwxProcessPoint objects are associated with **GwxText** objects to create alphanumeric display fields and data entry fields. A Process Point is used to show and/or download the value of an associated signal (analog, digital, or string). **GwxProcessPoint** has all the properties and methods of **GwxDynamic**.

Properties

ConfirmBeforeWrite

```
ConfirmBeforeWrite As Boolean
```

TRUE to prompt the user with a confirmation message before writing data values.

UseKeypad

```
UseKeypad As Boolean
```

TRUE to use an onscreen number pad or keyboard for this data entry.

Methods

Function GetLastEnteredValue

```
Function GetLastEnteredValue()
```

Returns the last entered value, which might be different from the currently displayed value.

GwxTimedate

GwxTimedate objects are associated with **GwxText** objects to create fields which display the current time and/or date. **GwxTimedate** has all the properties and methods of **GwxDynamic**.

GwxPoint

GwxPoint is the object type that GraphWorX32 uses to represent data connections. If several **GwxDynamic** objects are connected to the same data source, they reference a single shared **GwxPoint** object. **GwxPoint** objects handle OPC tags, expressions, constant values, and GraphWorX32 local variables. **GwxPoint** objects cannot be explicitly created or destroyed. GraphWorx32 automatically manages the lifetimes of **GwxPoint** objects based on the data source connections of the **GwxDynamic** objects in the display.

Properties

dataType

```
dataType As GWXDATATYPE
```

Data type of this point object. Valid values for **GWXDATATYPE** are:

```
DataTypeEmpty = 0
DataTypeShort = 2
DataTypeLong = 3
DataTypeFloat = 4
DataTypeDouble = 5
DataTypeString = 8
DataTypeBool = 11
DataTypeChar = 16
DataTypeByte = 17
DataTypeWord = 18
DataTypeDword = 19
```

highRange

```
highRange As Variant
```

High range value associated with this point object.

lowRange

```
lowRange As Variant
```

Low range value associated with this point object.

Value

```
Value As Variant
```

Current data value of this point object. This property gets updated with new values during runtime mode.

Methods

Function GetObjectTypeName

```
Function GetObjectTypeName() As String
```

Returns the type name of this object.

Function GetPointName

```
Function GetPointName() As String
```

Gets the point name (data source string) of this point object.

GraphWorX32 VBA Project

The GraphWorX32 VBA project is loaded whenever you launch the VBA Editor from the GraphWorX32 application by pressing **ALT+F11.** This project contains groups of modules by default, such as:

- GraphWorX32 Native Objects
- Modules - *ThisDisplay* module and *GwxTools* module
- Forms

Each module can contain VBA code - functions, subroutines, event handlers, and global declarations.

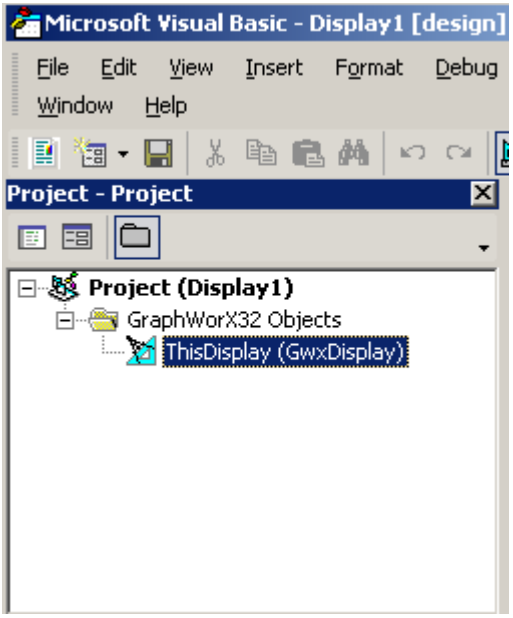
For more information about VBA, please see the GENESIS32 VBA Tutorial or the Microsoft VBA Help documentation.

GwxTools Module

GwxTools is a custom module with common subroutines used in the VBA Wizard. The GwxTools module is not available to the user until a VBA Wizard is used.

ThisDisplay Module

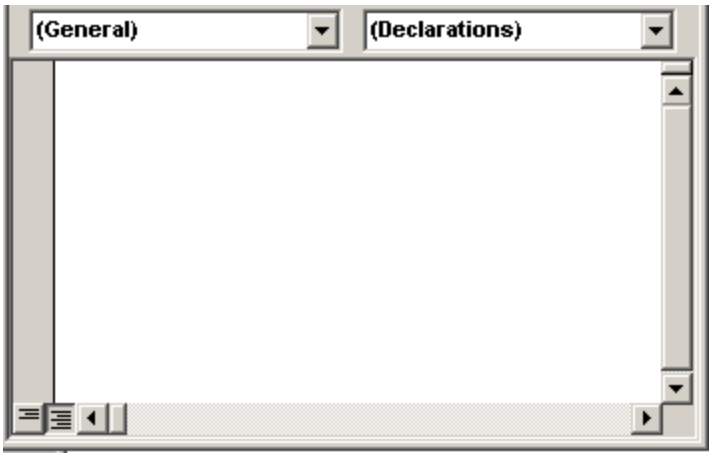
ThisDisplay is a special module representing the current GraphWorX32 display.



Project Window Displaying *ThisDisplay* Module

To display the code for *ThisDisplay* module, open the **Project Explorer** from the **View** menu of the VBA Editor. Expand the **GraphWorX32 Objects** group and double-click the *ThisDisplay (GwxDisplay)* item.

A *ThisDisplay* code window opens as below:



***ThisDisplay* Code Window**

ThisDisplay contains two **combo boxes** at the top.

The top left combo box allows you to select items such as:

- (General)
- ***GwxDisplay***

An empty display has only the above two items. If you have ActiveX controls in the display, the ActiveX items will also be displayed in this combo box.

The top right combo box shows events for the left combo selection. For example, the ***GwxDisplay*** item has events such as ***DisplayLoad*****,** ***DisplayUnload*****,** and others.

Event Handling With the "ThisDisplay" Module

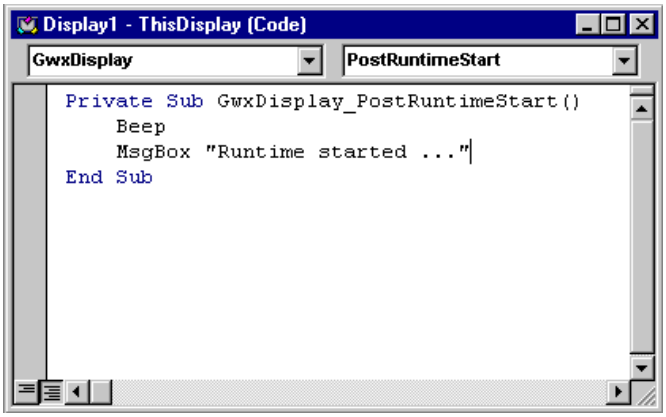
Event handlers can be added for a display using VBA. To do this, you should have the display loaded on the screen.

1. Open the VBA Editor by selecting **Macros > Visual Basic Editor** from the GraphWorX32 **Tools** menu. The GraphWorX32 VBA Project will be loaded by default.
2. Open the ***ThisDisplay*** module from the VBA Editor.
3. Select the ***GwxDisplay*** module in left combo box of ***ThisDisplay*** module.
4. Select the desired event from the list in the right combo box.
5. The subroutine header is automatically inserted into the current module. Insert your VBA code to the body of the subroutine.
6. Save the GraphWorX32 Project by selecting **Save Display 1** from the **File** menu.
7. Exit the VBA Editor by selecting **Close and Return to GraphWorX32** from the **File** menu.

Example

Write code for popping-up a message box that will display the message "Runtime Started" when you set your display into runtime.

1. Select the ***GwxDisplay*** module.
2. Select the ***PostRuntimeStart*** event from the list in the right combo box.
3. Insert VBA code, as shown in the figure below, into the body of the subroutine in the subroutine header:



PostRuntimeStart Event Example

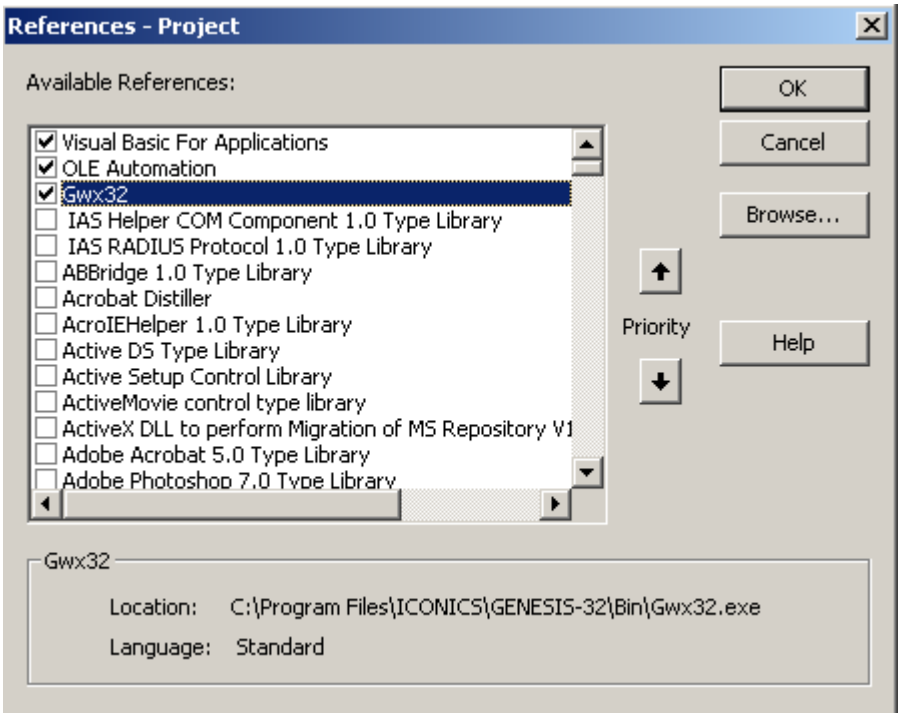
4. Switch to GraphWorX32 either by closing the VBA Editor or by pressing the **ALT+TAB** keys.
5. Test the event by starting runtime by clicking **Runtime** on the GraphWorX32 menu bar or by pressing the **CTRL+M** keys.
6. You should hear a beep, and a message box with a message "**Runtime started**" should appear.

How to Control GraphWorX32 Automation Using VBA

The VBA programmer has full control over the properties and native objects of the current display and can control the rich animation interface of GraphWorX32.

The Automation properties and methods are accessible through the *ThisDisplay* module. *ThisDisplay* is, as we said above, a representation of a current GraphWorX32 display and contains all Automation properties and methods.

Note: VBA can display Automation methods and properties only if the Automation object is checked in the **References** dialog, which can be opened from the menu bar. Select **References** from the VBA Editor **Tools** menu. Note that GraphWorX32 is checked automatically for every new display, as shown in the figure below.



References Dialog

How to Access GraphWorX32 Native Display Objects

GraphWorX32 native objects, such as ellipses, rectangles, symbols and dynamic actions, are not exposed to VBA by default. However they can be referenced from VBA if they have an *Object Name* assigned through the Property Inspector dialog.

A reference (actually a dispatch pointer) to a named object can be then retrieved by one of following methods of the ***ThisDisplay, symbol,*** and ***Visible*** objects.

- ThisDisplay.GetDynamicObjectFromName
- ThisDisplay.GetVisibleObjectFromIndex
- ThisDisplay.GetVisibleObjectFromName
- Visible.GetDynamicObjectFromIndex
- Visible.GetDynamicObjectFromName
- symbol.GetVisibleObjectFromIndex
- symbol.GetVisibleObjectFromName

ThisDisplay.GetVisibleObjectFromName("*Name*"). This function takes the name of a visual object in the display and reports back the object associated with that name. Upon storing the result of this function in a variable, you can change the properties of the object in the display by changing the fields of the variable. The following example rotates a rectangle named "Square" 45 degrees from horizontal.

```
Dim obj As GwxRectangle
Set obj = ThisDisplay.GetVisibleObjectFromName("Square")
obj.Angle = 45
```

ThisDisplay.GetDynamicObjectFromName("*Name*"). This function takes the name of a dynamic object in the display and reports back the dynamic object associated with that name. A dynamic object is not a physical, observable object but an abstraction for the operation performed on a visible object (i.e. Hide, Rotation, Size). Once the dynamic object is stored in a variable, you can change its properties. The following example changes the data source of a GraphWorX32 Hide object to zero, hiding the visible object associated with the dynamic object.

```
Dim obj As GwxHide
Set obj= ThisDisplay.GetDynamicObjectFromName("hd")
obj.dataSource = 0
```

ThisDisplay.GetVisibleObjectFromIndex(*Long Value*). This function selects an object based on the order in which visible objects on the screen were created. (The first visible object put on the screen has an index of 0.) This function is useful for iterating through all the objects in a display. The following code turns the first object created to green and the second to red.

```
Dim obj1 As Object
Dim obj2 As Object
Set obj1 = ThisDisplay.GetVisibleObjectFromIndex(0)
Set obj2 = ThisDisplay.GetVisibleObjectFromIndex(1)
obj1.fillColor = RGB(0, 255, 0)
obj2.fillColor = RGB(255, 0, 0)
```

***SymbolName*.GetVisibleObjectFromIndex(*Long value*)** Each GraphworX32 symbol has its own index that keeps track of the objects within it. The GetVisibleObjectFromIndex method, when appended to the name of a symbol, finds the visible object within the symbol with the specified index. This function is useful for iterating through all objects in a symbol. The following example turns the third visible object in a symbol named "sym" to green.

```
Dim sym1 As GwxSymbol
Dim obj As Object
Set sym1 = GetVisibleObjectFromName("sym")
Set obj = sym1.GetVisibleObjectFromIndex(2)
obj.fillColor = RGB(0,255,0)
```

(Note that the third object has index of 2. The first has index 0.)

***SymbolName*.GetVisibleObjectFromName("*Name*")** Finds a visible object within a symbol by the object name specified in GraphWorX32.

***VisibleObjectName*.GetDynamicObjectFromIndex(*Long value*)** Every time a dynamic object is assigned to a unique visible object, it is assigned an index. The first dynamic object assigned is given an index of zero. Since one visible object can be associated with many dynamic objects, this function provides a

useful way of manipulating dynamic objects. The following code takes the second dynamic object associated with a rectangle named "rect" and changes its low range to 10.

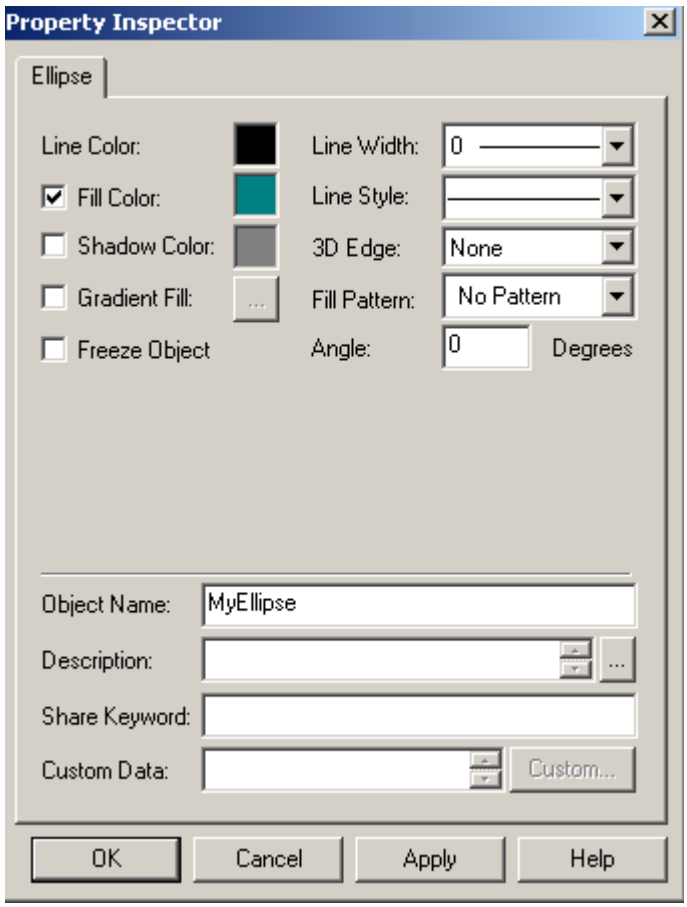
```
Dim o_Vis As GwxRectangle
Dim o_Dyn As Object
Set o_Vis = ThisDisplay.GetVisibleObjectFromName("rect")
Set o_Dyn = o_Vis.GetDynamicObjectFromIndex(1)
o_dyn.lowRange = 10
```

***VisibleObjectName.*GetDynamicObjectFromName("*Name*")** This function takes the name of a dynamic object associated with a visible object and allows you to represent the dynamic object with a variable.

Example

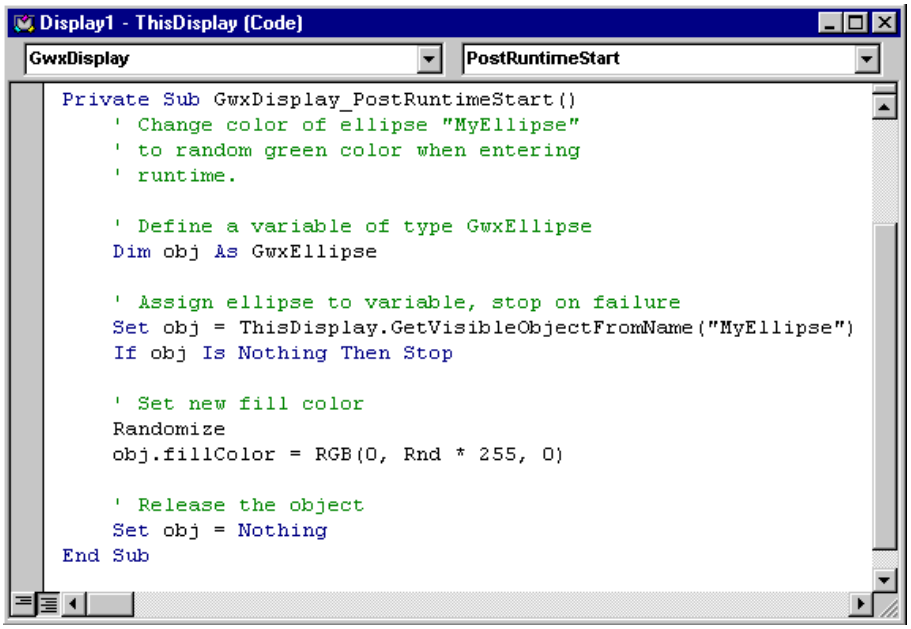
Write code for changing the color of an ellipse GraphWorX32 native object during Runtime.

1. Load an existing display or create a new GraphWorX32 display.
2. Draw an ellipse by clicking **Ellipse** on the **Draw** toolbar.
3. Double-click on the Ellipse. The **Property Inspector** dialog box opens, as shown in the figure below. Type the name **MyEllipse** in the object name field and then click on **OK.**
4. Open the VBA Editor by pressing **Alt+F11.**



Ellipse Object

5. Select the **ThisDisplay** module.
6. Select the **GwxDisplay** module in left combo box of ThisDisplay module.
7. Select the **PostRuntimeStart** event from the list in the right combo box.
8. The subroutine header is automatically inserted into the current module. Insert the VBA code into the body of the subroutine, as shown in the figure below.



GraphWorX32 Ellipse Modified From VBA

9. Save the GraphWorX32 project by selecting **Save Display 1** from the File menu.
10. Test the example by switching to the GraphWorX32 display and going into runtime.
11. The color of the ellipse should change to a random green color.

Object Names Must Be Unique

Sometimes it is desired to access the same objects in all duplicates (clones) of the desired symbol in the same way. This technique is useful, for example, for VBA Wizards, which have a macro behind the symbol. This macro is shared by all duplicates of the symbol and can be run on any of these duplicates.

To allow this feature, GraphWorX32 supports so called 'partial' names for objects in symbols. The partial name is a name that ends with an underscore '_', e.g. MyEllipse_. The duplicates of MyEllipse_ are then MyEllipse_1, MyEllipse_2, etc. What is important is the symbol method GetVisibleObjectFromSymbol(partialName), which accepts this partial name and returns the first occurrence of the specified object in specified symbol.

Example 1

Assume we have a symbol that consists of a rectangle and an ellipse. We want to write code that modifies these objects in any copy of the symbol. We must assign partial names to both objects (let's name them MyRect_ and MyEll_). Here is the code that shows how to access these objects in a specific symbol:

```
Dim sym As GwxSymbol
Set sym = FindSomehowDesiredSymbol()
' user method to choose the symbol
Dim ell As GwxEllipse, rect As GwxRectangle
Set ell= sym.GetVisibleObjectFromName("MyRect_")
```

```
Set rect= sym.GetVisibleObjectFromName("MyEll_")
' do something with these objects
' release references
Set ell = nothing
Set rect = nothing
Set sym = nothing
```

Example 2

The following example demonstrates the above technique.

1. In GraphWorX32, create one rectangle, and give it the Object Name **Rect_1.**
2. Duplicate this rectangle to create one more rectangle. Notice that this is automatically named **Rect_2.**
3. Group Rect_1 and Rect_2 together.
4. Duplicate this group using the duplicate button on the **Draw** toolbar.
5. Ungroup the duplicate object.
6. Read the Object Name by using the Property Inspector dialog. Notice that the Object names of the two rectangles (the third and the fourth) are **Rect_3** and **Rect_4.**

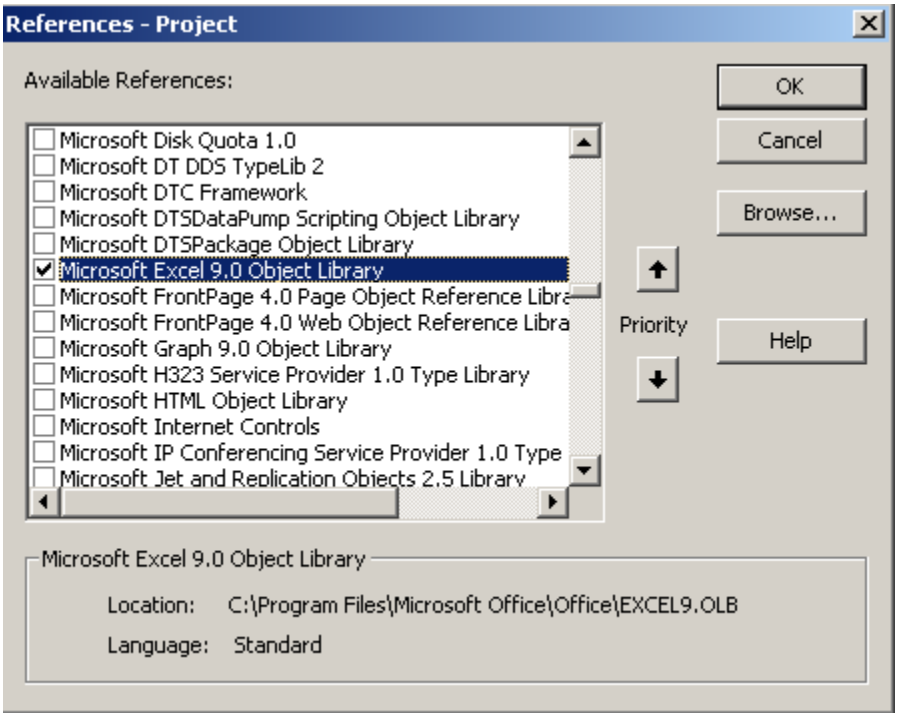
Using VBA to Connect With Other Applications

One powerful feature of VBA is that it allows you to link to other Windows applications and to exchange data. In the following example, you will see how GraphWorX32 can send data to, and receive data from, an Excel worksheet by using VBA.

Example

Design a display and a spreadsheet, each with two data points, and have them communicate to each other through VBA.

1. Open a new GraphworX32 project and go to the VBA Editor by pressing **ALT+F11.**
2. The first thing you need to do is to get the VBA in GraphworX32 to recognize Microsoft Excel data types. In the Visual Basic Editor, select **References** from the **Tools** menu. This opens a list of available References to applications. Check the box next to "Microsoft Excel Object Library," as shown in the figure below.

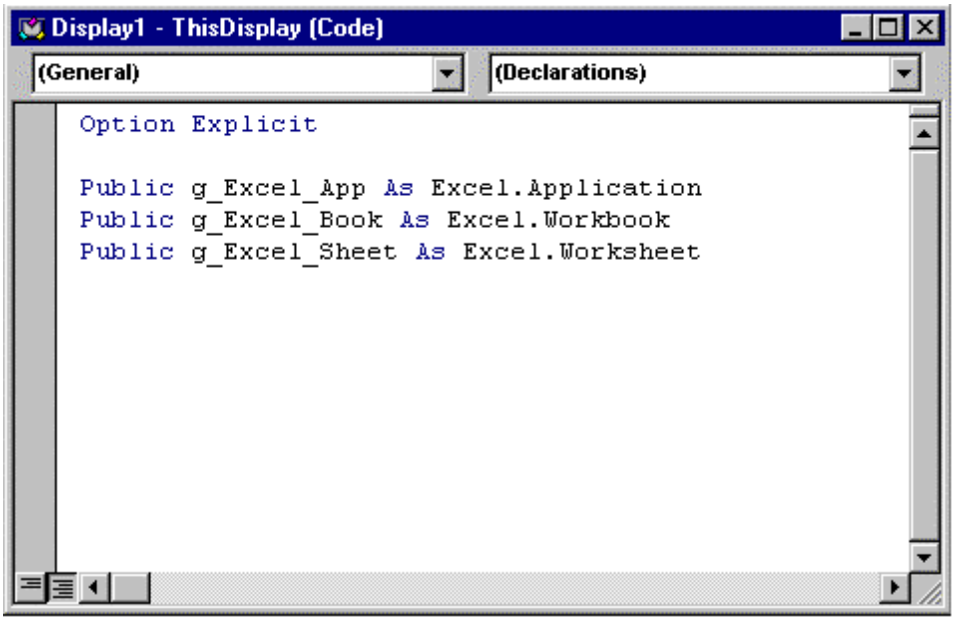


Selecting Microsoft Excel in the References Dialog

3. In the Project viewer window, double click on the ***ThisDisplay*** Module. You should see a code window with the words "Option Explicit." In that window, you want to make some global declarations that will be used later, as shown in the figure below:

```
Option Explicit

Public g_Excel_App As Excel.Application
Public g_Excel_Book As Excel.Workbook
Public g_Excel_Sheet As Excel.Worksheet
```

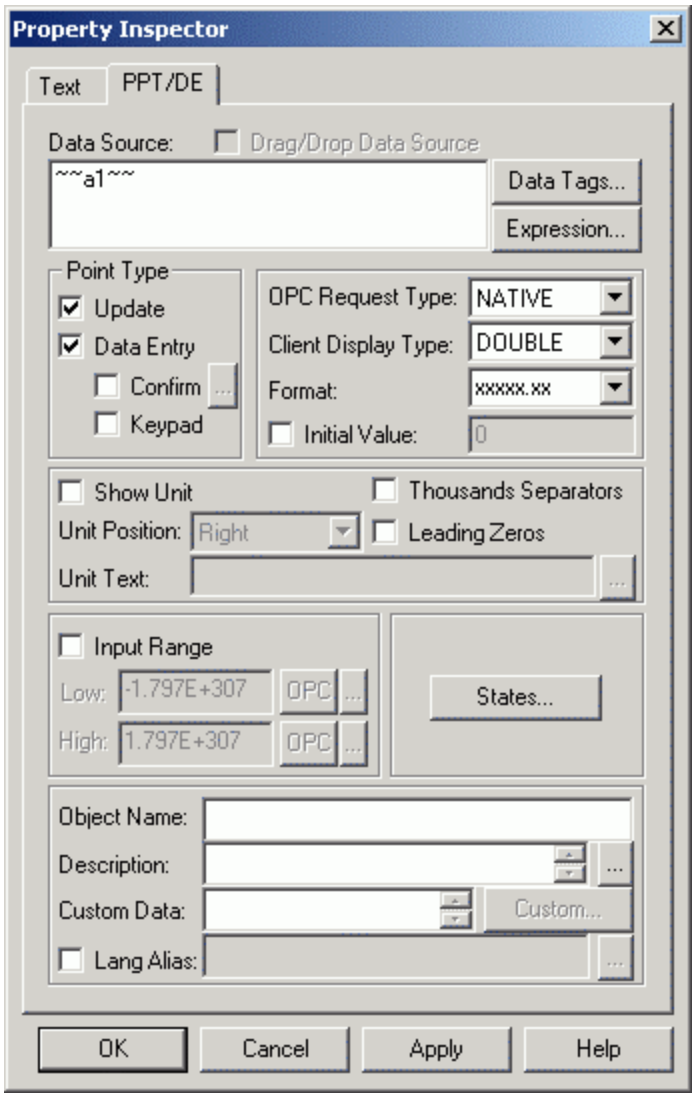


ThisDisplay Code Window With Global Declarations

4. In the left combo box at the top of the code window, select ***GwxDisplay***. In the right combo box, select ***PreRuntimeStart.*** By entering the following code, you will cause Excel to be launched before GraphWorX32 goes into runtime.

```
Private Sub GwxDisplay_PreRuntimeStart()
  ' Open up Excel and make it visible
  Set g_Excel_App = CreateObject("Excel.application")
  g_Excel_App.Visible = True
  ' Open up a sheet
  Set g_Excel_Book = g_Excel_App.Workbooks.Add
  Set g_Excel_Sheet = g_Excel_Book.Worksheets(1)
  ' Initialize the two cells you will be using
  g_Excel_Sheet.Range("a1") = 0
  g_Excel_Sheet.Range("a2") = 0
End Sub
```

5. Create two process points in your GraphWorX32 display. Make sure, for both process points, that the **Data Entry** check box is checked, as shown in the figure below. Connect one to a local variable ~~a1~~ and the other to a local variable named ~~a2~~. This double tilde notation is a standard notation for local variables in GraphWorX32.



Configuration of Process Point

6. Now go back to the Visual Basic Editor. You will now make a new procedure designed to read from and write to Excel. Select **Procedure** from the **Insert** menu. Name the procedure "Read_Write" and then enter in the following code.

```
Public Sub Read_Write(ByVal Co As Integer)
' This procedure reads from or writes to excel
' based on the condition, Co, passed as a
' parameter

  Dim Point As GwxPoint
  Dim St As String  'Used to store cell name
  Dim St2 As String 'Used to store variable name
  Dim X As Integer
  For X = 1 To 2
    ' add "a" to the value X converted to string
```

```
    St = "a" + Mid(Str(X), 2, 1)
    St2 = "~~" + St + "~~"
    Set Point=ThisDisplay.GetPointObjectFromName(St2)
    If Co = 1 Then  'Write to Excel
      ThisDisplay.g_Excel_Sheet.Range(St)=Point.Value
    Else          'Read From Excel
      Point.Value=ThisDisplay.g_Excel_Sheet.Range(St)
    End If
  Next X
End Sub
```

7. Return to the GraphWorX32 display and create two radio buttons. Name one of them "Read from Excel" and the other "Write to Excel." Configure both of them to run a VBA Script. The execution trigger should be "while down." Create a new macro for each of the two radio buttons. Call one macro "Rd" and call the other "Wr." Both of these two macros will call the "Read_Write" procedure. The code for each macro should look something like this:

```
Sub Wr(o As GwxPick)
  Call ThisDisplay.Read_Write(1)
End Sub

Sub Rd(o As GwxPick)
  Call ThisDisplay.Read_Write(0)
End Sub
```

8. Go back to your GraphWorX32 display and go into runtime. You will notice that Excel starts up automatically with the value 0 in the "A1" and "A2" cells. If you click on the "Write to Excel" button and change the value of one of the process points in GraphWorX32, you will notice the value in one of the cells in Excel changes. If you click on the "Read from Excel" button and change one of the two values in Excel, the process point will then update to match the value in Excel.

Other Sources of Information

For VBA programming, working with the modules and forms, and customization of the VBA Editor, refer to Microsoft Visual Basic Programmer's Guide or documentation on VBA and the help file that comes with VBA. You can open it from the VBA Editor **Help** menu.

Note that the VBA in GraphWorX32 is same as the one used in Microsoft Office applications (Word, PowerPoint, Access, Excel), and other products. Once you master VBA in GraphWorX32, you are able to program in all other applications.

There are many examples included with GraphWorX32, which are good sources of information and VBA programming tips and tricks.

It is also possible and helpful to open two or more instances of GraphWorX32, in each instance open the VBA Editor, and copy and paste the VBA code between the instances.

For more information about VBA, please see the GENESIS32 VBA Tutorial or the Microsoft VBA Help documentation.

Language Server OLE Automation Reference

The Language COM Server is the heart of the Language-Switching utility. It services requests from the connected client programs. The main purpose of the language server is to find translation parameters for given alias names. The language server configuration data (translation parameters, alias names, and supported languages) are stored in a database (Microsoft Access, MSDE or Microsoft SQL Server).

The language server is designed as an out-of-process COM object that supports two kinds of interfaces: Custom interface and Dual interface.

OLE Automation Objects and Interfaces

Microsoft Visual Basic uses the **IcoLangServer** Type Library to define the following interfaces. Make sure that, in the Visual Basic Project References, **ICONICS Language Server 1.0 Type Library** is checked. This library contains the following two objects:

- LangServer
- LangServerRecVarDescriptor

LangServer Object

Description	A language server client creates the LangServer automation object. The LangServer object can be used to obtain language translations and general information about a language server.
Syntax	LangServer
Example	Dim ALangServer As LangServer Set ALangServer = New LangServer

Summary of Properties

AliasStartDelimiter	ConfigurationDSrc	LoggingEnabled
AliasStopDelimiter	LoggerDSrc	VariableText

Summary of Methods

GetTranslationText	GetTranslationRecordset	GetLocaleID

LangServer Properties

AliasStartDelimiter

Description	(Read-only) Returns the opening delimiter string for translation alias. The LangServer object uses this string to identify the beginning of the alias string in an input string.
Syntax	AliasStartDelimiter As String
Remarks	This property, as well as other properties, is stored in the language-configuration database. An error occurs if the server has not been able to connect to the configuration database.
Example	Dim AnAliasStartDelimiter As String AnAliasStartDelimiter = AnLangServer.AliasStartDelimiter

AliasStopDelimiter

Description	(Read-only) Returns the ending delimiter string for translation alias. The LangServer object uses this string to find out the end of the alias string in an input string.
Syntax	AliasStopDelimiter As String
Remarks	This property, as well as other properties, is stored in the language-configuration database. An error occurs if the server has not been able to connect to the configuration database.
Example	Dim AnAliasStopDelimiter As String AnAliasStopDelimiter = AnLangServer.AliasStopDelimiter

ConfigurationDSrc

Description	(Read-only) Returns the file path that LangServer object is using to connect to the configuration database. It can be the file path to a MS Access database file (extension .mdb) or to a MS data link file (extension .udl). Data link file is a text file that contains an OLE DB connection string.
Syntax	ConfigurationDSrc As String
Remarks	This property is stored in the computer registry database. No errors should occur.
Example	Dim LangServerConfigurationDataSource As String LangServerConfigurationDataSource = AnLangServer.ConfigurationDSrc

LoggerDSrc

Description	(Read-only) Returns the file path that the LangServer object is using to connect to the message log database. It can be the file path to a MS Access database file (extension .mdb) or to a MS data link file (extension .udl). Data link file is a text file that contains an OLE DB connection string.
Syntax	LoggerDSrc As String
Remarks	This property is stored in the computer registry database. No errors should occur.
Example	Dim LangServerLoggerDataSource As String LangServerConfigurationDataSource = AnLangServer.LoggerDSrc

LoggingEnabled

Description	(Read-only) When TRUE, then the LangServer object logs information and warning messages to the database specified in the LoggerDSrc property. If FALSE, then the server does not log any message to the logger database.
Syntax	LoggingEnabled As Boolean
Remarks	This property, as well as other properties, is stored in the language-configuration database. An error occurs if the server has not been able to connect to the configuration database.
Example	Dim ALoggingEnabled As Boolean ALoggingEnabled = ALangServer.LoggingEnabled

VariableText

Description	(Read-only) Returns the expression variable identification string. This string identifies the input value in the expression string. The language server client should replace all occurrences of this string in an expression string with "real" input value and then evaluate it using the expression engine.
Syntax	VariableText As String
Remarks	This property value is hard-coded in the LangServer object. In future versions, it will be stored in the configuration database.
Example	Dim AVariableText As String AVariableText = ALangServer.VariableText

LangServer Methods

GetTranslationText

Description	Returns the text translation(s) for a specified AliasID string.
Syntax	GetTranslationText (AliasID As String, Language As String, LanguageSubset As String, ClientIdentification As String, ErrMessageLocaleID As Long, FontFaceName As String) As String

Part	Description
AliasID	The AliasID string contains alias name(s) for which the caller wants to read the text translation(s). For the AliasID string "any text *Alias1* any other text *Alias2* etc…" the result will be "any text *TextTranslation1* any other text *TextTranslation2* etc…".
Language	The Language name for which the caller wants to read the translation(s).
LanguageSubset	The Language subset name for which the caller wants to read the translation(s). It can be an empty string if the caller wants to read translation for default language subset.
ClientIdentification	The LangServer object uses the ClientIdentification string to identify message records in the logger database. It should contain a unique client identification (including e.g. display name) and node information.
ErrMessageLocaleID	The ErrMessageLocaleID parameter is the locale ID that the LangServer object will use to create an error message description (method IErrorInfo::GetDescription(…..)).
FontFaceName	The FontFaceName returns font face name associated with the first alias string in AliasID parameter. The client should use the font face name to display translated text. If the FontFaceName is an empty string, then the client uses its default settings.

Remarks	This method can fail from many reasons. Three basic reasons are: **1.** The LangServer object has not been able to connect the configuration database. **2.** AliasID string contains invalid delimiters (opening delimiter was found but not ending, etc.). **3.** Language or Language Subset string does not exist in the configuration database.
Example	Private Sub cmdSomeButton_Click()

```
        On Error GoTo L_ERROR
        ' Read translation text for two aliases with names
"Alias1" and "Alias2";
        ' delimiters are `` and ''

        Dim ALangServer As LangServer                    '
Declare Alias Server
        Set ALangServer = New LangServer                 '
Create an instance
        If ALangServer Is Nothing Then          ' Check for
failure
                MsgBox "Failed to create an instance of
'LangServer'", vbOKOnly, "ICONICS LangServer"
                Exit Sub
        End If

        Dim FontName As String
        Dim Text As String

        ' Get translation for language "English - United
States" and default language subset; error
        ' messages in English
        strText =
ALangServer.GetTranslationText("``Alias1'' ``Alias2''",
"English - United States", "", "My identification", 1033,
strFontName)

        ' Some code to show the results (strText,
strFontName)….
        Set ALangServer = Nothing
        Exit Sub

L_ERROR:                                         ' Handle
errors
        LblTranslText.Caption = ""
        MsgBox "Failed to obtain alias: " & Err.Description,
vbOKOnly, "ICONICS AliasServer"
End Sub
```

GetTranslationRecordset

Description	Returns the LangServerRecVarDescriptor object that contains all translation parameters (text translation, read/write expressions, etc.) for a specified AliasID string.
Syntax	GetTranslationRecordset (AliasID As String, Language As String, LanguageSubset As String, ClientIdentification As String, ErrMessageLocaleID As Long, Descriptor As LangServerRecVarDescriptor)

Part	Description
AliasID	The AliasID string has to contain alias name for which the caller wants to read the translation parameters. This method returns

	translation parameters for the first found alias name only (compare with GetTranslationText method).
Language	The Language name for which the caller wants to read the translation parameters.
LanguageSubset	The Language subset name for which the caller wants to read the translation parameters. It can be an empty string if the caller wants to read translation parameters for default language subset.
ClientIdentification	The LangServer object is using the ClientIdentification string to identify message records in the logger database. It should contain unique client identification (including e.g. display name) and node information.
ErrMessageLocaleID	The ErrMessageLocaleID parameter is the locale ID that the LangServer object will use to create an error message description (method IErrorInfo::GetDescription(…..)).
LangServerRecVarDescriptor	The GetTranslationRecordset method is using this object to return all translation parameters (see chapter LangServerRecVarDescriptor for details).

Remarks	This method can fail from many reasons. Four basic reasons are: **1.** The LangServer object has not been able to connect the configuration database. **2.** AliasID string contains invalid delimiters (opening delimiter was found but not ending, etc.). **3.** AliasID string does not contain any translation alias name. **4.** Language or Language Subset string does not exist in the configuration database.
Example	Private Sub cmdSomeButton_Click() On Error GoTo L_ERROR ' Read translation parameters for alias with name "Alias1"; delimiters ' are `` and '' Dim ALangServer As LangServer ' Declare Alias Server Set ALangServer = New LangServer '

```
Create an instance
        If ALangServer Is Nothing Then          ' Check for
failure
                MsgBox "Failed to create an instance of
'LangServer'", vbOKOnly, "ICONICS LangServer"
                Exit Sub
        End If

        Dim RecDescriptor As LangServerRecVarDescriptor
        ' Declare descriptor object


        ' Get translation parameters for language "English -
United States" and
        ' default language subset
        Call ALangServer.GetTranslationRecordset
("``Alias1"", "English - United States", "", "My identification",
1033, RecDescriptor)

        ' Add some code to show the results from
RecDescriptor:
        ' RecDescriptor.TranslationText,
RecDescriptor.ExpressionFormat, etc.

        Set ALangServer = Nothing
        Exit Sub

L_ERROR:                                        ' Handle
errors
        LblTranslText.Caption = ""
        MsgBox "Failed to obtain alias: " & Err.Description,
vbOKOnly, "ICONICS AliasServer"
End Sub
```

GetLocaleID

Description	Returns the locale ID associated with specified Language name.
Syntax	GetLocaleID (Language As String, ErrMessageLocaleID As Long) As Long

Part	Description
Language	The Language name for which the caller wants to read the locale ID.
ErrMessageLocaleID	The ErrMessageLocaleID parameter is the locale ID that the LangServer object will use to create an error message description (method IErrorInfo::GetDescription(…..)).

Remarks	This method can fail from many reasons. Two basic reasons are: **1.** The LangServer object has not been able to connect the configuration database. **2.** Language string does not exist in configuration

	database,
Example	Private Sub cmdSomeButton_Click() On Error GoTo L_ERROR Dim ALangServer As LangServer ' Declare Alias Server Set ALangServer = New LangServer ' Create an instance If ALangServer Is Nothing Then ' Check for failure MsgBox "Failed to create an instance of 'LangServer'", vbOKOnly, "ICONICS LangServer" Exit Sub End If Dim LanguageLCID As Long ' Get LCID for language "English - United States" LanguageLCID = ALangServer.GetLocaleID ("English - United States", 1033) ' Add some code to show the result Set ALangServer = Nothing Exit Sub L_ERROR: ' Handle errors LblTranslText.Caption = "" MsgBox "Failed to obtain alias: " & Err.Description, vbOKOnly, "ICONICS AliasServer" End Sub

LangServerRecVarDescriptor Object

Description	The LangServerRecVarDescriptor object is a collection of translation properties. All supported properties are read-only. The LangServer object creates the LangServerRecVarDescriptor object and initializes it with translation parameters (see GetTranslationRecordset method in LangServer object). The client then can read all these properties to get translation parameters.
Syntax	LangServerRecVarDescriptor
Example	Dim ADescriptor As LangServerRecVarDescriptor

Summary of Properties

TranslationText	ExpressionFormat	ReadExpression
WriteExpression	FontFaceName	

LangServerRecVarDescriptor Properties

TranslationText

Description	(Read-only) Returns the translation text for a specified alias.
Syntax	TranslationText As String
Remarks	The TranslationText value is created similarly to the result of GetTranslationText method (LangServer object) but it returns translation text for the first found alias only.
Example	Dim ATranslationText As String ATranslationText = ADescriptor.TranslationText

ExpressionFormat

Description	(Read-only) Returns the expression format string associated with a specified alias. This format string should be used to format numeric (expression results) or date values. If this string is empty then no formatting should be done. The format string is using following syntax: 1. Numeric formats: "TYPE=<*type_name*> FORMAT=<*format_string*>" where *type_name* can be one of following: FLOAT, DOUBLE, BOOL, BYTE, WORD, DWORD, CHAR, SHORT, LONG and *format_string* is the string that uses "x" characters to specify numeric or decimal places, "." character specifies decimal point. 2. Date-time formats: "TYPE=<DATETIME> TIME=<*time_format_string*> DATE=<*date_format_string*>". DATE and TIME keywords are optional; if the formats string contains both then their

	order specifies the order of time and date values in the result.
Syntax	ExpressionFormat As String
Example	Dim AnExpressionFormat As String AnExpressionFormat = ADescriptor.ExpressionFormat

ReadExpression

Description	(Read-only) Returns the read expression string associated with specified alias. The client uses this expression to convert read value for a specified language (or language subset) translation. If the ReadExpression contains an empty string, then no conversion is necessary.
Syntax	ReadExpression As String
Remarks	The expression string contains special substring(s) defined by *VariableText* property in LangServer object. This substring should be replaced by input value, and then this expression string can be evaluated using the expression engine .dll.
Example	Dim AReadExpression As String AReadExpression = ADescriptor.ReadExpression

WriteExpression

Description	(Read-only) Returns the write expression string associated with a specified alias. The client uses this expression to convert a written value for a specified language (or language subset) translation. If the ReadExpression contains an empty string, then no conversion is necessary.
Syntax	WriteExpression As String
Remarks	The expression string contains special substring(s) defined by *VariableText* property in LangServer object. This substring should be replaced by the input value, and then this expression string can be evaluated using the expression engine .dll.
Example	Dim AWriteExpression As String AWriteExpression = ADescriptor.WriteExpression

FontFaceName

Description	(Read-only) Returns the font face name associated with a specified alias. The client should use the font face name to display translated text. If the FontFaceName is an empty string, then the client uses its default settings.
Syntax	FontFaceName As String
Example	Dim AFontFaceName As String AFontFaceName = ADescriptor.FontFaceName

Recipe ActiveX OLE Automation Reference

This reference describes the OLE Automation features available in the Recipe ActiveX. The Recipe ActiveX provides a COM interface that allows automation interfaces run from within the ActiveX container to manipulate the Recipe ActiveX control as it is running. The interface is available to all programming languages that support COM, including Visual Basic (VB), Visual Basic for Applications (VBA), and Microsoft Visual C^{++}.

To access the Automation interface from VB and VBA, the Recipe ActiveX must be made available by selecting **RecipeLib** in the Object Browser.

Control Properties

AutoSelectRow

Type: Boolean

Description

If true, when the user clicks on a cell the whole row will be highlighted.

BackColor

Type: OLE_COLOR

Description

Sets/gets the background color of the column.

Example

```
Dim column As OGridColumnWrapper
Set column = ThisDisplay.Recipe1.GetColumn(1)
column.ModifyBackColor = True
column.BackColor = RGB(0, 255, 0)
```

The above example will change the background color for column 1 to green.

BorderType

Type: String

Description

Sets/gets control's border appearance. Can be one of the following:

"0" - none

"1" - flat border

"2" - 3D border

Example

Sets 3D look for the control's border.

```
ThisDocument.Recipe1.BorderType="2"
```

CellHAlign

Type: Short

Description

Sets/gets the default cell alignment.

“0” – Left Alignment

“1” – Center Alignment

“2” – Right Alignment

Example

ThisDocument.Recipe1.CellHAlign="2"

ColumnHAlign

Type: Short

Description

Sets/gets the column header alignment.

“0” – Left Alignment

“1” – Center Alignment

“2” – Right Alignment

Example

ThisDocument.Recipe1. ColumnHAlign ="1"

DisplayGridTitle

Type: Boolean

Description

Sets/gets Boolean value that controls the visibility of the grid title bar.

Example

ThisDocument.Recipe1. DisplayGridTitle = TRUE

FileName

Type: BSTR

Description

Sets/gets the name of the Recipe ActiveX configuration file.

Example

ThisDocument.Recipe1.FileName=”C:\RecipeConfig.rcp”

Font

Type: StdFont

Description

Returns a reference to the control's default font.

Example

To set control's default font to Arial, 14 pt., use the following statements:

```
With ThisDocument.Recipe1.Font
.Name="Arial"
.Size=14
End With
```

ForeColor

Type: OLE_COLOR

Description

Sets/gets foreground color (cell’s font color) property of the Recipe ActiveX.

Example

```
ThisDocument.Recipe1. ForeColor=RGB(0,0,0)
```

GridBackColor

Type: OLE_COLOR

Description

Sets/gets color value used to paint the gaps between rightmost column and right edge of the grid, last row and bottom edge of the grid, and row headers column (the 'back area' of the grid).

Example

```
ThisDocument.Recipe1. GridBackColor =RGB(0,127,0)
```

GridDefRowHeight

Type: Integer

Description

Sets/gets the default row height for data grid.

Example

```
ThisDocument.Recipe1. GridDefRowHeight=16
```

GridHeaderShade

Type: Short

Description

Sets/gets the cell effect for the grid‘s title bar header.

"0" = Soft Bump

"1" = Diagonal Shade

"2" = Horizontal Shade

"3" = Vertical Shade

"4" = Horizontal Bump

"5" = Vertical Bump

"6" = Metal

GridHorizLinesColor and GridVertLinesColor

Type: OLE_COLOR

Description

Sets/gets color for horizontal or vertical grid lines.

Example

ThisDocument.Recipe1. GridHorizLinesColor =RGB(127,127,127)

ThisDocument.Recipe1. GridVertLinesColor = RGB(127,127,127)

GridHorizLinesStyle and GridVertLinesStyle

Type: Integer

Description

Sets/gets style for horizontal or vertical grid lines. Can be one of the following:

0 - no line

1 - solid line

2 - dot line

3 - dash line

4 - dashdot line

5 - dashdotdot line

Example

ThisDocument.Recipe1. GridHorizLinesStyle =0

ThisDocument.Recipe1. GridVertLinesStyle = 2

GridHorizLinesWidth and GridVertLinesWidth

Type: Integer

Description

Sets/gets the widths for the horizontal or vertical grid lines when corresponding grid line style (property GridHorizLinesStyle or GridVertLinesStyle) is set to 1 - solid line. For all other line styles, these properties must be set to 1.

Example

Sets both horizontal and vertical grid lines to solid line style, 2 pixels wide.

ThisDocument.Recipe1. GridHorizLinesStyle =1

ThisDocument.Recipe1. GridVertLinesStyle = 1

ThisDocument.Recipe1. GridHorizLinesWidth =2

ThisDocument.Recipe1. GridVertLinesWidth = 2

GridShade

Type: Short

Description

Sets/gets the cell effect for the grid column‘s headers.

“0” = Soft Bump

“1” = Diagonal Shade

“2” = Horizontal Shade

“3” = Vertical Shade

“4” = Horizontal Bump

“5” = Vertical Bump

“6” = Metal

GridTitleBackColor

Type: OLE_COLOR

Description

Sets/gets the background color for the grid's title bar header.

Example

Sets title bar background color to dark blue.

ThisDocument.Recipe1. GridTitleBackColor =GRB(0,0,127)

GridTitleFont

Type: StdFont

Description

Returns a reference to the default font for the grid’s title bar header.

Example

To set control's default font to Arial, 14 pt., use the following statements:

With ThisDocument.Recipe1.Font

.Name="Arial"

.Size=14

End With

ThisDocument.Recipe1. GridTitleFontColor =GRB(0,0,127)

GridTitleHAlign

Type: Short

Description

Sets/gets the grid's title bar header alignment.

"0" – Left Alignment

"1" – Center Alignment

"2" – Right Alignment

Example

ThisDocument.Recipe1. GridTitleHAlign ="1"

GridTitleTextColor

Type: OLE_COLOR

Description

Sets/gets the text color for the grid's title bar header.

Example

Sets grid's title bar text color to yellow.

ThisDocument.Recipe1. GridTitleBackColor =GRB(255,255,0)

HeaderColor

Type: OLE_COLOR

Description

Sets/gets the background color for grid's column headers.

Example

Sets color of grid's column header to dark green.

ThisDocument.Recipe1. GridTitleBackColor =GRB(0,127,0)

HeaderFont

Type: StdFont

Description

Returns a reference to the default font for the grid's column headers.

HeaderTxtColor

Type: OLE_COLOR

Description

Sets/gets the text color for grid's column headers.

Example

Sets grid's title bar text color to yellow.

ThisDocument.Recipe1. GridTitleBackColor =RGB(255,255,0)

HideGridPopupMenu

Type: Boolean

Description

Enables/disables Recipe ActiveX runtime popup menu.

InternalGridEdges

Type: Short

Description

Sets the amount of space between the ActiveX borders and the grid borders.

RecipeClientID As String

Type: BSTR

Description

Sets/gets the Recipe Identification string used for binding to the tree explorer tag.

Example

ThisDocument.Recipe1.RecipeClientID=”Recipe1”

RecipeTitle As String

Type: BSTR

Description

Sets/gets the title shown in the window title bar of the Database Mining ActiveX control during runtime mode.

Example

ThisDocument.Recipe1.RecipeTitle=”Grid Title”

ShowColsHeader and ShowRowsHeader

Type: Boolean

Description

Show/hide the rows/columns headers.

URLPathName

Type: BSTR

Description

Sets/gets the URL path for a configuration file over the Web. Must set URLPathUsed to “True” for this to take effect.

Example

ThisDocument.Recipe1.URLPathName=
”http://www.iconics.com/recipecfg.rcp”

URLPathUsed

Type: Boolean

Description

Enables/disables the use of a URL path.

Example

ThisDocument.Recipe1.URLPathUsed=True

UseLocalSettings;

Type: Boolean

Description

When the URL path is enabled, it allows you to use or not to use the local settings configuration file.

Example

ThisDocument.Recipe1.UseLocalSettings=True

Control Methods

AboutBox

Shows **About** dialog box.

Example

ThisDocument.Recipe1.AboutBox

GetColumnsNumber As Integer

Returns the number of columns currently present in the gird.

Return Value

On success, returns the number of columns; if it fails or the grid doesn’t have any column, returns 0.

GetGridCellValue(ColNum As Long, RowNum As Long)

Returns the value of the cell that has coordinates (ColNum,RowNum).

Return Value

On success, returns the value of the cell; if the cell coordinates are not valid or if the grid is not visible on the screen, it will fail returning an empty string.

GetRowsNumber As Integer

Returns the number of rows currently present in the grid.

Return Value

On success, returns the number of rows; if it fails or the grid does not have any columns, returns 0.

GetSelectedColIndex As Integer

Returns the coordinates of the currently selected column.

Return Value

On success, returns the coordinates of the currently selected column; if it fails, returns 0.

GetSelectedRange(VARIANT* TopCrd, VARIANT* LeftCrd, VARIANT* BottomCrd, VARIANT* RightCrd)

Returns the top, left, right and bottom coordinates of the currently selected area in the grid.

Return Value

On success, returns the coordinates of the currently selected area in the grid; if it fails there is no selected area, and returns all the coordinates with a negative value.

GetSelectedRowIndex As Integer

Returns the coordinates of the currently selected row.

Return Value

On success, returns the coordinates of the currently selected row; if it fails, returns 0.

LoadConfigFile() As Boolean

It loads the configuration file specified in newVal.

Example

Dim ret As Boolean

ret = Recipe1.LoadConfigFile(“C:\My Configs\cfg.rcp”)

Function LoadConfigURL() As Boolean

Loads the configuration file specified in the URL string newVal.

Example

Dim ret As Boolean

ret = Recipe1.LoadConfigURL(“http://localhost/cfg.rcp”)

Function ReplaceHost() As Long

It works over data source tags, recipe tags and path name attributes and replaces node name **substring** within URL path only. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces host name in whole display

Dim Status As Long
Status = ThisDisplay.ReplaceHost("Host1", "Host2")
'if Status <> 0 then there was no replacement performed or an error occurs
If Status <> 0 Then
 MsgBox "No replacements"
Else
 MsgBox "Tags replaced"
End If
```

Function ReplaceHostEx() As Long

It works over data source tags, recipe tags and path name attributes and replaces node name **substring** within URL path only, and supports case-sensitivity, wildcard strings and MatchWholeWord flag. Returns 0 on success and HRESULT when something fails.

Example

```
'similar to ReplaceTag
'VBA example, works only in configure mode
'replaces host name in whole display, regarding "case" and "whole words" options
```

Function SaveConfigFile() As Boolean

It saves your current Recipe ActiveX configuration to the file specified in newVal.

Example

```
Dim ret As Boolean
ret = Recipe1.SaveConfigFile(“C:\My Configs\cfg.rcp”)
```

Sub UnselectAllRows()

Unselects the currently selected area if present.

Example:

```
ThisDisplay.Recipe1.UnselectAllRows
```

Control Events

Event OnGridLButtonClicked(nCol As Integer, nRow As Long)

Description

This event is called when the user clicks on any grid's cell giving the cell's coordinates (nRow,nCol).

Event OnGridLButtonDblClk(nCol As Integer, nRow As Long)

Description

This event is called when the user double clicks on any grid's cell giving the cell's coordinates (nRow,nCol).

Event OnGridRButtonClicked(nCol As Integer, nRow As Long)

Description

This event is called when the user right clicks on any grid's cell giving the cell's coordinates (nRow,nCol).

Note: this event will be fired up only if HideGridPopupMenu property is set to false

Event OnGridRButtonDblClk(nCol As Integer, nRow As Long)

Description

This event is called when the user right double clicks on any grid's cell giving the cell's coordinates (nRow,nCol).

Note: this event will be fired up only if HideGridPopupMenu property is set to false

Event OnGridSelectionChanged(bIsDragging As Integer, bKeyPressed As Integer)

Description

This event is called when the user start to drag the mouse inside the grid's cell area, or a range of cells, rows or columns has been selected, or if the current selected range has been modified.

bIsDragging will be set to true if the user is still dragging and when he will release the mouse this event will be called once more with bIsDragging set to false.

bKeyPressed specifies if the user has selected the cells by pressing SHIFT and an arrow key.

Event OnUpdate()

Description

This event is called every time the grid's values are refreshed

ScriptWorX32 OLE Automation Reference

The ScriptWorX32 Engine has an automation interface that will be available to any other automation application, such as GraphWorX32. The following is a list of methods that are allowed to control and use ScriptWorX32.

ScriptWorX32 has set of inter-threaded communication objects, which allows you to create your VBA scripts simply and more stable. These objects are very important to the current ScriptWorX design. Every VBA script runs in separated thread, and these VBA scripts needs to communicate somehow.

- **Swx32GlobalStorage** object

  It is a set of objects that allows storing and manipulating data between different VBA script threads.

- **IQueue** object

  This object implements data queueing capability. One or more VBA scripts are getting data from the queue and the other threads are putting them into. It is a key object to create scripts with thread safe data processing. Typical usage is:
  Create one ScriptWorX “infinite” VBA script triggered as global script at the beginning and implement data processing into it (e.g. data logging capability with all the database connections, sending emails, etc.)

  Then create one or more VBA scripts driven by periodic events or alarm events, which will put its data into the queue object. These scripts will end up as fast as possible to prevent ScriptWorX threads blocking.

- **ICounter** object

  Implements thread-safe counter with upper and lower boundaries.

- **IDataPoint object**

  Allows easy OPC data access (it is using GenBroker/GenClient support to access OPC servers).

- **IObjectVariable**

  Thread safe automation pointers (IDispatch) distribution.

- **IVariable**

  Thread safe update or exchange of VARIANT variables.

- **Swx32Synchronization** object

  This object contains a set of two thread synchronization objects.

- **IEvent** object

  Thread events with timeouts – one thread is waiting for setting event from other thread.

- **ILock** object

  Thread locking with locking timeout.

- Property **ThreadsRunning** as Long

  The number of threads currently running.

- Property **LoggerLCID** as Long

  The ID number of the logger.

- Property **MessageLCID** as Long

  The ID number of the message.

- Function **CreateVariable**(Name as String)

  Creates a global variable and initializes it to zero.

- Function **DestroyVariable**(Name as String)

  Destroys the specified global variable.

- Function **GetServerTime**(pLowDateTime as Long, pHighDateTime as Long)

  Gets the current time.

- Function **GetVariable**(Name as String) as Long

  Gets the content of a global variable.

- Function **PrintToConsole**(Message as String)

  Prints the message to the ScriptWorX32 console.

- Function **ResumeScript**(ScriptID as Long)

  Resumes script instance ScriptID. ScriptID was returned from StartScript() call in output parameter.

- Function **SetVariable**(Name as String, newVal as Variant)

  Sets a global variable to the specified value.

- Function **StartScript**(ScriptName as String, Project as String, Module as String, StrGUID as String, ScriptID as Long) as Long

  Starts instance of ScriptName from VBA MT library identified by a DllProgID. The DLL library must be compiled and registered in the VBA Editor. The output parameter ScriptID is used in subsequent calls to identify the script.

- Function **StopThreads**()

  Stops all running threads.

- Function **SuspendScript**(ScriptID as Long)

  Suspends script instance ScriptID. ScriptID was returned from StartScript() call in output parameter.

- Function **TerminateScript**(ScriptID as Long)

  Terminates script instance ScriptID by a soft method. ScriptID was returned from StartScript() call in output parameter. Note this is dangerous operation that can cause loss of data and resource leaks.

- Function **TerminateAllScripts**()

  Terminates all running instances of scripts by a soft method.

- Function **TerminateThreads**()

  Terminates all running threads by a soft method.

- Function **VariableExists**(Name as String) as Boolean

  Returns true if the specified global variable exists.

- Function **Alarm32_SetEvents**(EventList as Variant)

  Sets events for Alarm32 Server.

- Function **Alarm32_GetEvents**(EventListPtr as Variant)

  Gets events from Alarm32 Server.

- Function **Alarm32_SetSubscriptions**(Subscriptions)

  Sets subscriptions for Alarm32 Server.

- Function **Event32_SetEvents**(EventList as Variant) as Long

  Sets events for Event32 Server.

- Function **Event32_GetEvents**(EventListPtr as Variant)

  Gets events from Event32 Server.

- Function **ExitServer**()

  Shuts down the server by closing the ScriptWorX32 visible client.

- Function **GetSecurity**(SecurityID as Long) as Boolean

  Gets security on the specified item.

- Function **Periodic32_SetEvents**(EventList as Variant)

  Sets events for Periodic32 Server.

- Function **Periodic32_GetEvents**(EventListPtr as Variant)

  Gets events from Periodic32 Server.

- Function **SetConsoleLoggerOptions**(ConsoleLoggerOptions as Long)

  Sets the ScriptWorX32 console logger options.

- Function **SetEventLoggerOptions**(EventLoggerOptions as Long)

  Sets the NT event logger options.

- Function **SetMonitorPtr**(pDispatch as Object)

  Set back-pointer where the SwxEngine notified events to.

- Function **SetWorkingDirectory**(WorkingDirectory as String)

  Sets the ScriptWorX32 Engine current working directory.

- Sub **StartRuntime**()

  Starts runtime mode in ScriptWorX32.

- Sub **StopRuntime**()

  Returns ScriptWorX32 to configuration mode.

All of these actions are called from the ScriptWorX32 user interface but may be used by any other application as well. This means any other application will be able to run and control scripts, obtain monitor notifications about running instances, programs the in-proc servers, and so on. Other applications should only need to use the StartScript() method.

ScriptWorX32 is registered at GenRegistrar and is thus available to all other applications.

Writing a Script

Once you have configured the triggering of the script, it is necessary to write the corresponding script itself. A script is a Visual Basic for Applications (VBA) public subroutine stored in the referenced project and module, as configured in the trigger options in the ScriptWorX32 user interface.

Note: ScriptWorX32 supports ScriptWorx32/VBA document synchronization and contains has VBA Script Wizards, which greatly simplify the creation of new scripts.

Note: It is strongly suggested that the configuration of a script in the ScriptWorX32 user interface and the writing of the script be conducted simultaneously. This will help to ensure that the Script, Project, and Module names match up correctly, and will also make testing the script and its configuration much easier.

1. Before writing the script, it is necessary to set the working directory, which is the directory where the compiled .dll will be located and where all configuration files will be saved. To set the working directory, select **Set Working Directory** from the **Tools** menu.

2. When you are ready to write a script, either click on the **Visual Basic Editor** button on the toolbar, or select **Macros - Visual Basic Editor** from the **Tools** menu. This will launch the Visual Basic Editor, where you will write the script.

3. Once in the Visual Basic Editor, make sure the project name and module correspond to the entries made in the ScriptWorX32 user interface. The script name will correspond to the name used in the actual code, as shown below.

```
Public Sub ScriptName()

        Msgbox "hello"

End Sub
```

Note: All scripts are written in Visual Basic and obey all rules and methods therein. For questions regarding the actual writing of the code, please refer to the Visual Basic documentation.

4. Once the code has been written, save the project by selecting **Save As** from the **File** menu, and enter the same project name as was referenced in the configuration of the script.

5. Compile the VBA file by selecting **Make ProjectName.DLL** from the **File** menu.

6. After the file has been compiled, close the Visual Basic Editor and return to ScriptWorX32.

7. Enter runtime mode and watch the scripts execute.

Examples

Several examples of ScriptWorX32 configurations are provided in the installation under "\Program Files\ICONICS\GENESIS32\Examples\ScriptWorX32". Except for the "Outlook Mail" and "Running Display" examples, each example can be configured using the text file that is associated with the example.

Carousel

The Carousel example opens a GraphWorX32 display file every time the script is run. The displays to be shown are listed in the text file "Carousel.txt," which is located in the ScriptWorX32 Examples Directory. Notice that the files listed do not have a path. The example reads a line from the file and pastes the path of GEN32DEMO in front of the line. This means that all displays that you want to add to the Carousel must be located in the GEN32DEMO directory.

To use this example, you must have ICONICS GraphWorX32 installed on your computer, as well as the GEN32DEMO Example. Make sure that the GWX Object is selected in the **Tools - References** menu in the VBA Editor.

Customer DB

The Customer DB example cannot be configured. Every time the **launchCustomerForm** method is called, a VBA form appears. Fill in this form and click the **Close** button. The data entered in the form will then be saved in a Microsoft Access Database "CustomerDb.mdb" file using the Microsoft DAO 3.51 Object.

Excel Logger

The Excel Logger example writes OPC tag values to Microsoft Excel using the Excel OLE automation interface. The Excel file and the OPC tags to be logged are stored in the text file "ExcelLogger.txt." The first line in the file is the location of the Excel File where the log should be written. The remaining lines are valid OPC tags.

Note: The module used to retrieve the OPC tag values only supports local OPC servers. To use this Example, you must have Microsoft Excel 97 or 2000 installed on your computer. Make sure that the Microsoft Excel Object (version 8.0 or 9.0) is selected in the **Tools - References** menu in the VBA Editor.

Outlook Mail

The Outlook Mail example writes OPC tag values to a Microsoft Outlook mail message. You can select the tags to log in the configuration form what appears when calling the **showConfiguration** method. In this form, you also can select the recipients and the subject of the message.

Every time the method **sendMail** is called, the example will open Microsoft Outlook, retrieve the current values of the selected OPC tags, and write these values in the e-mail message. When these tasks are completed, the message is sent and Microsoft Outlook is closed.

To use this example, you must have Microsoft Outlook installed on your computer, and the Outlook Object must be selected in the **Tools - References** menu in the VBA Editor.

Word Logger

The Word Logger example writes OPC tag values to Microsoft Word using the Word OLE automation interface. The Word document and the OPC tags to be logged are stored in the text file "WordLogger.txt." The first line in the file is the location of the Word Document where the log should be written. The remaining lines are valid OPC tags.

Note: The module used to retrieve the OPC tag values only supports local OPC servers. To use this example, you must have Microsoft Word 97 or 2000 installed on your computer. Make sure that the Microsoft Word Object (version 8.0 or 9.0) is selected in the **Tools - References** menu in the VBA Editor.

Script Configuration Examples

The following are detailed examples of how to configure scripts using ScriptWorX32.

Beep

The first example is very simple and is designed as a test to make sure that ScriptWorX32 is communicating properly between all of its parts. With the help

of Script Wizard and automatic document synchronization, the creation of new script trigger and script itself is an easy task.

Example 1

1. Open a new ScriptWorX32 file.
2. Insert a new periodic script by right-clicking **Periodic Scripts** on the tree view in the left-hand pane and selecting **Insert Trigger.** Alternatively, select **Insert New Trigger** from the **Edit** menu, or simply click the **Insert Trigger** button on the toolbar.
3. Save the configuration file under desired the name, e.g. Sample.swx.
4. Check that the Project and Module/Designer fields are automatically filled in for you. The Project corresponds to the VBA project, and the Module corresponds to the VBA designer module.
5. Select the **Interval** tab, and make sure that the 1 second interval is already entered.
6. In the configuration screen, change the name to **doBeep** in the **Trigger Name** and **Script Name** fields. The script name corresponds to the name of the script placed in VBA Editor in the related Project and Module.
7. Click **Apply.**
8. Click the **Edit VBA Code** button. The VBA Editor opens, and the script skeleton is already created there:

```
Public Sub DoBeep()

  ' TODO: Add your procedure code here

End Sub
```

9. Enter the body of the script:

```
    Beep
```

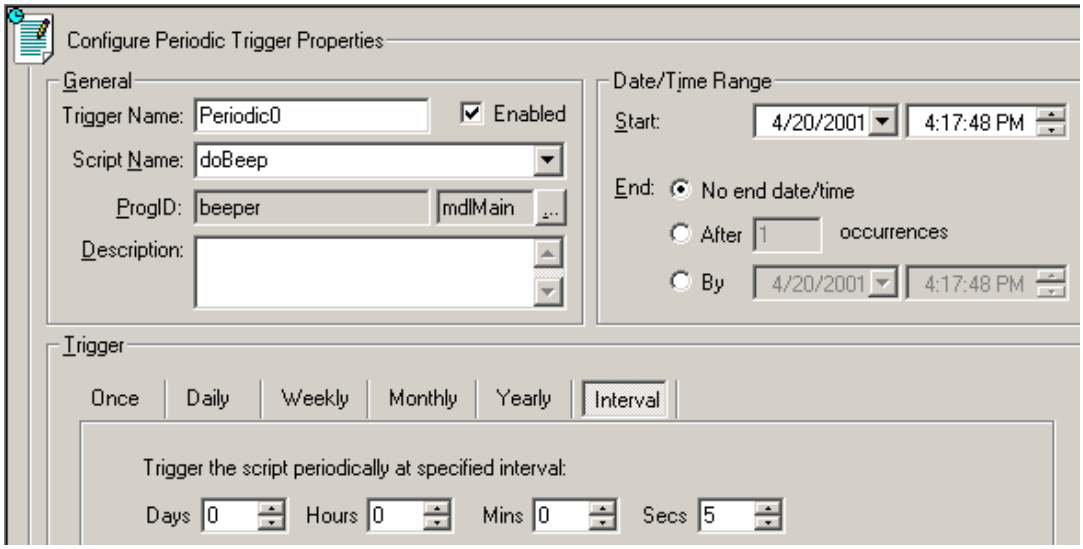
10. Compile the project into .dll by selecting **Make Sample.dll** from the **File** menu in the **VBA Editor.**
11. Click the **Runtime** menu in ScriptWorX32.

Example 2

1. Open a new ScriptWorX32 file in the user interface.
2. Insert a new periodic script as described above.
3. Enter the following information into the proper fields for this new script, as shown below.

Field:	**Entry**

Script Name:	doBeep
Project:	beeper
Module/Designer:	mdlMain
Start Date/Time:	Current
End	No end date/time
Trigger Interval:	5 sec



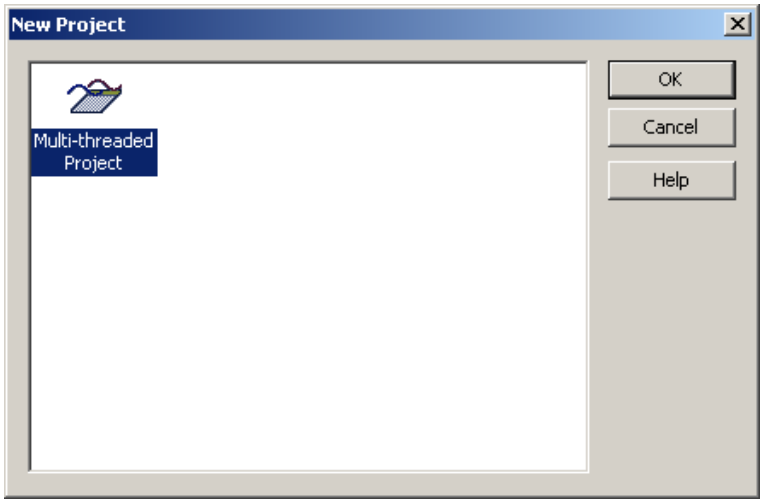
doBeep Configuration

4. It is also necessary to set the working directory. This will be the directory where the compiled .dll file will be located, as should all configuration files directly related to this project. To set the working directory select **Set Working Directory** from the **Tools** menu.

5. Once the script has been configured as shown above, save the configuration file by selecting **Save As** from the **File** menu. Enter **Beeper.swx** as the file name.

6. Before this script is ready to be run, it is necessary to actually write the script. To do so, click the **Visual Basic Editor** button on the toolbar, or select **Macros - Visual Basic Editor** from the **Tools** menu.

7. In the Visual Basic Editor, open a new project by selecting **New Project** from the **File** menu. The **New Project** dialog box will open, as shown below.



New Project Dialog Box in VBA Editor

8. Select **Multi-threaded Project** from the dialog box.

9. In the **Properties** window of the VBA Editor, change the name of the module from Designer1 to mdlMain. (**Note:** this is the same module name used in the configuration of the script.)

10. From the Project Explorer window, select the mdlMain object and enter the following code:

```
Public Sub doBeep()

   Beep

End Sub
```

Notice that the script name, doBeep, is used in the code. This is how the script will be able to run properly.

11. At this point the script is effectively written. It is now necessary to save and compile the project .dll and save the project. First save the project by selecting **Save As** from the **File** menu. Name the file **beeper.vba.** Make sure the file is saved into the working directory that you set earlier.

12. Now compile the project .dll by selecting **Make beeper.DLL** from the **File** menu.

13. Once the .dll has been compiled, close the Visual Basic Editor by selecting **Close and return to ScriptWorX32** from the **File** menu.

14. Now it is possible to run the script from the user interface. To run the script, either select **Runtime!** from the menu bar, or click the **Runtime** button on the toolbar. This will launch the Runtime Monitor, which will show all scripts that are running, as well as each instance of each script that has run.

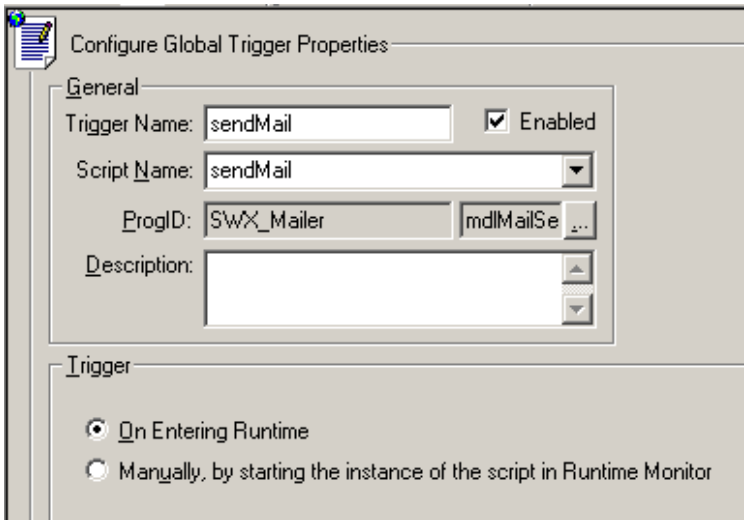
The script we have constructed in this example will beep every five seconds. If this does not occur during runtime, check your VBA code and your script configuration to make sure that the Script, Project, and Module names are consistent throughout.

Send Mail

This next example will start an instance of Microsoft Outlook, open a blank message file, address the message, and send the message.

1. To start off, insert a global script with the following properties.

Field:	**Entry**
Script Name:	sendMail
Project:	SWX_Mailer
Module/Designer:	mdlMailSender
Start Date/Time:	Current
End	No end Date/Time
Trigger	On Entering Runtime



Send Mail Script Configuration

2. Once you have set up the configuration according to the specified properties save the configuration by selecting **Save As** from the **File** menu. Enter **SWX_Mailer.swx** as the file name.

3. Now enter the code behind the script. To do so, start the Visual Basic Editor.

4. To use this example, you must have Microsoft Outlook installed on your computer, and the Outlook Object must be selected in the **Tools - References** menu in the VBA Editor.

5. Change the name of the module to **mdlMailSender,** and enter the following code in the code body window.

```
Public Sub sendMail()

   MsgBox "Before"

   Send_An_Email "test@iconics.com", "", "", "myTest", "Hello from
ScriptWorX"

   MsgBox "after"

End Sub


Sub Send_An_Email(P_to As String, P_cc As String, p_bcc As String,
subject As String, m_text As String)

On Error GoTo errHandler

Dim OutApp      As Outlook.Application

Dim OutMail     As Outlook.MailItem

Dim y           As Integer
```

```
    Dim bodytext    As String    'text of e-mail
    Dim Str_to_send As String 'value to send
      Set OutApp = CreateObject("Outlook.application")
      Set OutMail = OutApp.CreateItem(olMailItem)
      OutMail.To = P_to
      OutMail.CC = P_cc
      OutMail.BCC = p_bcc
      OutMail.subject = subject
      OutMail.Body = m_text
        OutMail.Send
        OutApp.Quit
  resetObjects:
      Set OutMail = Nothing
      Set OutApp = Nothing
      Exit Sub

    errHandler:
       MsgBox Err.Description, vbCritical, Err.Number
       GoTo resetObjects
    End Sub
```

6. Once the code is entered, save the code and compile the .dll.
7. After the .dll has been successfully created, close the Visual Basic Editor and return to ScriptWorX32.
8. Now it is possible to run the script from the ScriptWorX32 user interface. To run the script, either select **Runtime!** from the menu bar, or click the **Runtime** button on the toolbar.
9. This will launch the Runtime Monitor, which will show all scripts that are running as well as each instance of each script that has run.

While this script is running, it will launch a message box with the label "Before" (this text can be change by replacing this entry in the code). Once the **OK** button is clicked in this box, the test message will be sent. After the message is sent, a similar message box with the label "After" will be displayed.

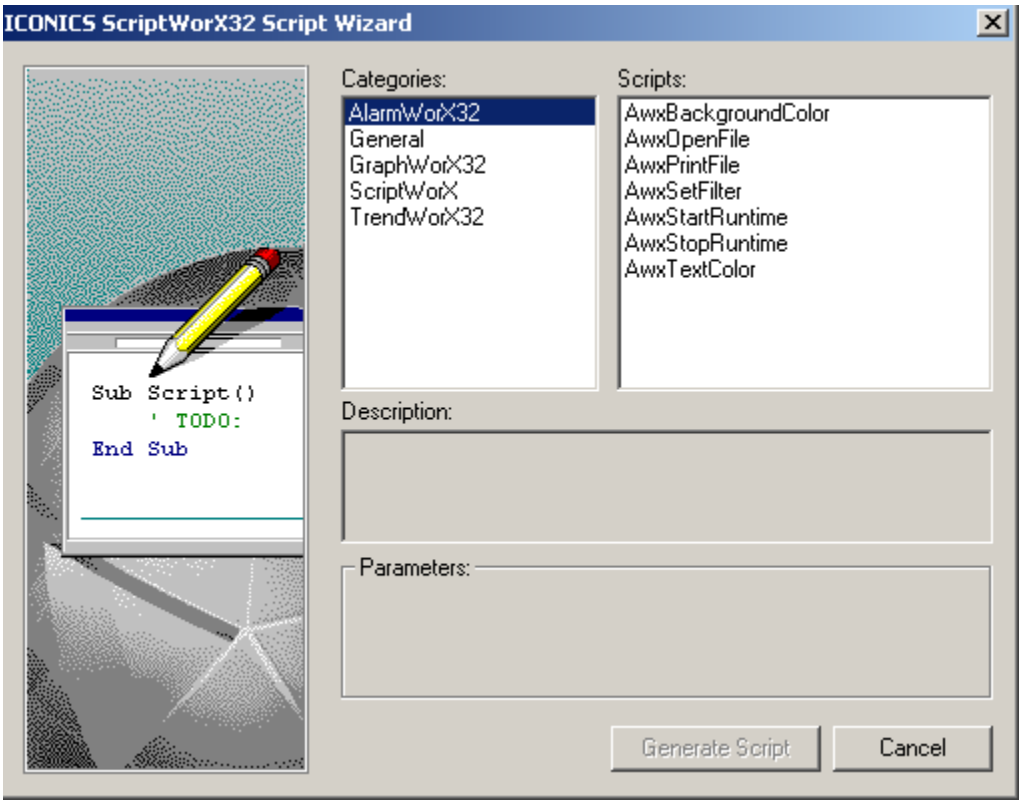
Note: To test to make sure this script is working correctly, enter your email address in place of test@iconics.com.

Script Wizard

The **Script Wizard** is a tool that is available from each script configuration window and that allows you to generate scripts from script templates. Each script template is stored in one script template file (.stp), which is located in the Script Wizard Template directory placed in the installation directory of ScriptWorX32. Script templates support several keywords, which are replaced during script generation by values entered by in the Wizard edit fields. Script templates also support the addition of necessary TypeLib references that can be appended to the VBA container. TypeLib references and optional keywords must be specified in the header section of the script.

Running the Script Wizard

To run the ScriptWorX32 VBA Script Wizard, insert a script configuration in the ScriptWorX32 user interface and click the **VBA Script Wizard** button on the configuration screen. Doing so will open the following window.



Script Wizard

The Script Wizard provides several categories of script templates from which to generate a script.

Description

The **Description** section tells you what the script is supposed to be doing and what components need to be installed on your system before the script will run properly. The text that is shown in this section is the text that was entered in the VBA code after the **#Description** keyword.

Parameters

Depending on the category and the script selected, several items are available in the **Parameters** section of the window. As stated in earlier sections, the parameters available are set in the configuration of the script template by using the **#Parameters** keyword. The entries for each of these fields will be directly inserted into the script and will determine exactly how the scripts run.

Installed Script Categories

There are five installed script template categories: AlarmWorX32, General, GraphWorX32, OPC Automation, and TrendWorX32. Each of these categories includes associated scripts. Each script includes a description of what the script does and what components must be installed.

AlarmWorX32

Script	Description
AwxBackgroundColor	Set background color of AWXView32 ActiveX. GraphWorX32 and AWXView32 must be installed.
AwxOpenFile	Open GraphWorX32 display with an AWXView32 ActiveX and load the AWXView32 configuration file.
AwxPrintFile	Print the display containing the AWXView32 ActiveX. GraphWorX32 and AWXView32 must be installed.
AwxSetFilter	Set filter name and filter expression for AWXView32 ActiveX. GraphWorX32 and AWXView32 must be installed.
AwxStartRuntime	Enter Runtime Mode. GraphWorX32 and AWXView32 must be installed.
AwxStopRuntime	Exit Runtime Mode. GraphWorX32 and AWXView32 must be installed.
AwxTextColor	Set text color of AWXView32 ActiveX. GraphWorX32 and AWXView32 must be installed.

General

Script	Description
Beeper	This script Beeps when started.
DiskSpace	The script obtains total and free disk space.
HelloICONICS	The script pops up a message box that says "Hello from ICONICS."
Keypad	The script displays a Keypad form for entering a text string.
Memory	The script obtains total and available physical memory.
Numpad	The script displays a Numpad form for entering a double value.

GraphWorX32

Script	Description
GwxArrayOfEllipses	Script starts GraphWorX32 and creates an array of ellipses, where the array dimensions and ellipse properties are specified.
GwxArrayOfRectangles	Script starts GraphWorX32 and creates an array of rectangles, where the array dimensions and rectangle properties are specified.
GwxClosePopup Window	Script starts GraphWorX32 and opens the popup window specified in the File Name field. Then the popup window is closed.
GwxEmbeddedWindow	Script starts GraphWorX32 and opens the display selected in the File Name field as an embedded window.
GwxIterateObjects	Iterates all symbols and their subsymbols in the display.
GwxMaximizeWindow	Script starts GraphWorX32 and maximizes its window. GraphWorX32 must be installed.
GwxMinimizeWindow	Script starts GraphWorX32 and minimizes its window. GraphWorX32 must be installed.
GwxOpenDisplay	Script starts GraphWorX32 and opens the display selected in the File Name field.

GwxPopupWindow	Script starts GraphWorX32 and opens a popup window specified in the File Name field. Use the Browse button.
GwxPrintDisplay	Script starts GraphWorX32, and opens and prints the display selected in the File Name field.
GwxSetBackgroundColor	Script starts GraphWorX32 and changes the display background color.
GwxSetDisplayDimensions	Script starts GraphWorX32 and sets the display dimensions (work area/world bounds) of the currently loaded display.
GwxSetWindowDimensions	Script starts GraphWorX32 and sets the GraphWorX32 main window size and location as a percentage of the total screen size.
GwxStartRuntime	Script starts GraphWorX32 and switches to runtime mode. GraphWorX32 must be installed.
GwxStopRuntime	Script starts GraphWorX32 and stops runtime mode.
GwxToggleRuntime	Script toggles GraphWorX32 runtime mode. GraphWorX32 must be installed.

ScriptWorX

Script	Description
Infinite Script	Implements body of an “infinite” script. It loops and waits for the ScriptWorX shut-down event.
Queue Source	Queue source implementation with error handling.
Queue Target	Queue target implementation with error handling – an infinite script with queue data reading.
ReadOPCTag	Read an OPC tag using IDataPoint object; waits for the first update.
ReadOPCTag_ NoWait	The same as “ReadOPCTag” script, but it does not wait for the first update.
WriteOPCTag	Write an OPC tag using IDataPoint object.

TrendWorX32

Script	Description
TwxAddTrend	Adds real-time trend to the TrendWorX32 Viewer. GraphWorX32 and TrendWorX32 Viewer must be installed.
TwxBackgroundColor	Sets TrendWorX32 Viewer background color. GraphWorX32 and TrendWorX32 Viewer must be installed.
TwxDeleteTrend	Deletes real-time trend pen. GraphWorX32 and TrendWorX32 Viewer must be installed.
TwxOpenFile	Script starts GraphWorX32 display with a TrendWorX32 Viewer ActiveX. GraphWorX32 and TrendWorX32 Viewer must be installed.
TwxPrintFile	Prints the display containing the TrendWorX32 Viewer ActiveX. GraphWorX32 and TrendWorX32 Viewer must be installed.
TwxShowDetails	Shows or hides the detailed list of trends. GraphWorX32 and TrendWorX32 Viewer must be installed.
TwxShowTitle	Shows or hides the TrendWorX32 Viewer title. GraphWorX32 and TrendWorX32 Viewer must be installed.
TwxStartRuntime	Enters runtime mode. GraphWorX32 and TrendWorX32 Viewer must be installed.
TwxStopRuntime	Exits runtime mode. GraphWorX32 and TrendWorX32 Viewer must be installed.

Using a Template

To access a script template, open the ScriptWorX32 user interface and click the **VBA Script Wizard** button. This will open the Script Wizard as described above.

1. Select the desired category and select the appropriate script from the list.
2. Click on the script. The description of the script should appear in the **Description** field.
3. Once you have set up the Wizard as desired, click the **Generate Script** button. The ScriptWorX32 user interface will be restored, and you will be asked if you want to view the VBA code.

Script Wizard Creation and Maintenance

This section is for more advanced users who wish to create their own script template files.

Header Section

The header section can contain optional keywords, which must be in the following format:

#KEYWORD: value

Optional keywords:

#REFERENCE: *module1*
#REFERENCE: *module2*
#REFERENCE: **…**

- Add *TypeLib* references to VBA .

#DESCRIPTION: *text1*
#DESCRIPTION: *text2*
#DESCRIPTION: **…**

- Enter script information that will be displayed in the Script Wizard dialog box.
- Description can be placed on several lines. Each of them must begin with the same keyword.

#PARAMETER: *parameter*

- *Parameter* may be one of the following strings.
- Pay attention to filling in this section; '***parameter'*** must be spelled correctly!
- Parameters are exclusive; i.e. only one of them can be used at a time.

par_FileName	- Script template includes file name parameter.
par_Tag	- Script template includes OPC tag.
par_None	- Has no effect.

The header section must be ended by a keyword:

#END

This marks the end of header section. It is the only compulsory header keyword. The other keywords are optional.

Optional and Required Parameters in Scripts

Optional parameters must be defined in the header section as described above. These parameters are one of the following:

par_FileName, par_Tag

Required parameters can be used without definition in the header section (because the related entry fields are always used in the Script Wizard dialog box). Required parameters are:

par_Name, par_Key, par_Node

When parameters are used in the script template code, they must be enclosed by **<<** and **>>** characters.

Parameters are replaced during script generation by the values specified by the user in the related edit fields.

par_FileName - When declared, the Script Wizard displays the edit field and **Browse** button to specify the file name.

par_Tag - When declared, the Script Wizard displays the edit field and **Browse** button to specify the OPC tag.

par_Name - Contains the script name.

par_Key - GenRegistrar key parameter.

par_Node - GenRegistrar node parameter.

Script Template Sample

```
#REFERENCE:     Gwx32
#DESCRIPTION:   Script starts GWX32 and opens display selected
#DESCRIPTION:   in the FileName field.
#DESCRIPTION:   Note Gwx32 must be installed.
#PARAMETER:     par_FileName
#END

' Script <<par_Name>> was generated by ICONICS Script Wizard
' from a template OpenDisplay.stp
'
Public Sub <<par_Name>>()
   ' Create an instance of GWX32
   Dim gwx As Gwx32.GwxDisplay
   Set gwx = New Gwx32.GwxDisplay

   If gwx Is Nothing then
      ' Report problem and exit
      MsgBox "GWX32 creation failed. Check it is installed and registered"
      Exit Sub
   End If

   call gwx.FileNew
   call gwx.FileOpen("<<par_FileName>>")
   call gwx.BringWindowToTop
```

```
    call gwx.ShowWindow

    MsgBox "Close GWX"
    call gwx.ExitApplication
    set gwx = nothing
End Sub
```

Example

In this example, we will create both a script wizard and a script as a result of this wizard, and will incorporate both the **#DESCRIPTION** and **#PARAMETER** keywords.

Building a Template

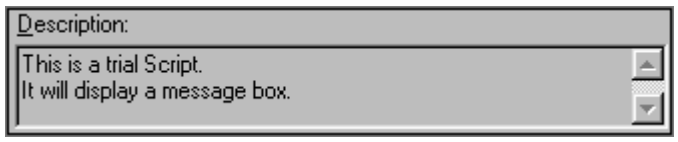
To start, open the **\Program Files\ICONICS\GENESIS32\Bin\ScriptWizard** directory. In this directory, you will find folders representing all of the above mentioned template categories.

1. Create a new category called **Tutorial** by creating a new folder in the Script Wizard directory and giving it the appropriate name.
2. Create a new .txt file and rename it **Tutorial.stp.**
3. You have now created a blank file that can be edited using notepad. Double-click on this newly created file.
4. You are now ready to start writing the code for your Script Wizard. To enter the information that will appear in the **Description** field, enter the following code:

   #DESCRIPTION: This is a trial script.

   #DESCRIPTION: It will display a message box.

The reason for having two different lines is that the **Description** field in the Script Wizard interface has a preset length and does not wrap text within the box. Once the Wizard is saved, check to make sure that all of your description text is visible to the user, as shown below:

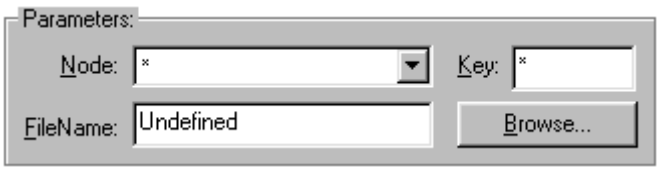


Description Section

5. Enter the following parameter keyword:

   #PARAMETER: par_FileName

Entering this parameter will add the FileName field to the Script Wizard, as shown below:



Parameter Section

Note: If the par_Tag parameter had been used, the FileName field would be attached to an OPC tag, and the **Browse** button would launch the Unified Data Browser.

6. Enter the end keyword: #END

This will indicate the end of the header section of the Wizard.

The code that makes the basis of the script is Visual Basic code, which references the different parameters set in the header. For our example, we will be creating a simple message box referencing a file selected by the user.

Note: The complexity of what a Script Wizard can accomplish is based solely on the extent to which the programmer is able to program in Visual Basic.

7. Enter the following code directly beneath the header section:

```
' Script <<par_Name>> (generated by the ScriptWizard from Tutorial)

Public Sub <<par_Name>>()

    MsgBox "What do you want to know about <<par_FileName>>!!!"

End Sub
```

The entry for <<par_Name>> is taken from the ScriptWorX32 user interface **Script Name** field. The entry for <<par_FileName>> is set in the Script Wizard user interface and is available since it was added in the header section.

After the script is coded as desired, save the file and close the instance of Notepad.

Using the Template

To access the script template that you have just created, open the ScriptWorX32 user interface and click the **VBA Script Wizard** button. This will open the Script Wizard as described above, but the Wizard should now contain an additional category called **Tutorial.**

Note: If this new category is not available, check to make sure that the folder containing the template is located in the same directory as the other template category folders.

1. Select the Tutorial category. The script template **Example** should be displayed in the scripts list.

2. Click on this script. The information you entered using the #Description keywords should appear in the **Description** field, and the **Parameters** section should contain the default **Node** and **Key** fields as well as the Optional **FileName** field.

3. To select a file name to be used in the script, click the **Browse** button and use the **Open** dialog to browse through available files.

4. Once you have set up the Wizard as desired, click the **Generate Script** button. This returns you to the ScriptWorX32 user interface. You will be asked if you want to view the VBA code.

Security OLE Automation Reference

The OLE Automation interface for the WebHMI Security ActiveX is compatible with VBA and VBScript. You can perform login/logout operations directly trough scripting without displaying any user interface. The WebHMI Security ActiveX contains the following OLE Automation interfaces:

LoginDlg()

Launches the login dialog.

ChangePwdDlg()

Launches the dialog to change the password.

ShowLoggedInUsers()

Launches the dialog to show a list of the usesr currently logged into the Security Server.

Logout()

Logs out all currently logged users.

SetTimeout(LONG nSec)

Sets the timeout for all of the GenClient calls to the Security Server.

ShowResultMsgs(BOOL bShow)

Enables / disables the message box with the result (e.g. "failed to log on to the Security Server").

LoginUser(BSTR username, BSTR password)

Logs in one specific user through code.

LogoutUser (BSTR username)

Logs out a specific user through code.

GetLoggedInUsersNames (BSTR usernames)

Gets the list of currently logged users. The string “usernames” is filled with the comma-separated list of currently logged user names. Note that by default the Security Server does not allow concurrent login of multiple users; the concurrent login option must be enabled from the Security Server Configurator. Please refer to the Security Server documentation for additional details.

Launching the Security Login ActiveX Through Scripting

The Security Login ActiveX can be programmatically created and initialized from VBA Script, VBScript and JScript. The GraphWorX32 Symbol Library contains a category named “WebHMI Security Login” located under the “VBAScriptSymbol” folder, which is filled with samples of each of the automation interfaces listed above. Please refer to the aforementioned samples for additional information on how to use the Login ActiveX through scripting.

The following code sample has been extracted from the Symbol Library; this sample shows how to launch the Login ActiveX from VBScript. The code runs on WebHMI too:

```
Set t = CreateObject ("ICONICS.LoginActiveX")
If t Is Nothing Then
 MsgBox "An error has occurred while trying to launch the login
dialog."
Else
 t.LoginDlg()
End If
```

The following code sample has been extracted from the Symbol Library; this sample shows how to get the list of currently logged users from VBScript. The code runs on WebHMI too:

```
Set t = CreateObject ("ICONICS.LoginActiveX")
If t Is Nothing Then
 MsgBox "An error has occurred while trying to launch the login
dialog."
Else
   t.GetLoggedInUsersNames str
 MsgBox str
End If
```

Symbol Library OLE Automation Reference

GraphWorX32 Display (configuration mode only)

- **Function InsertLibraryObjectByIndex(category As String, index As Long) As Boolean**

```
If Not ThisDisplay.InsertLibraryObjectByIndex("Misc", 0) Then
  MsgBox "Error with InsertLibraryObjectByIndex!"
Else
  Msgbox "Library object successfully inserted!"
End If
```

- **Function InsertLibraryObjectByName(category As String, symbol As String) As Boolean**

```
If Not ThisDisplay.InsertLibraryObjectByName("Misc", "Check")
Then
  MsgBox "Error with InsertLibraryObjectByName!"
Else
  Msgbox "Library object successfully inserted!"
End If
```

- **Function InsertLibraryObject(libraryObject As Object) As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Misc")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

Dim symbol As Object
Set symbol = sc.GetSymbolFromName("Check")
If symbol Is Nothing Then
  MsgBox "Failed to get symbol from name!"
  Exit Sub
```

```
End If

If Not ThisDisplay.InsertLibraryObject(symbol) Then
  MsgBox "Error with InsertLibraryObject!"
Else
  Msgbox "Library object successfully inserted!"
End If
```

- **Function ConvertGwxSymbolToLibraryObject(gwxSymbol As Object, libraryObject As Object) As Boolean**

```
Dim newSym As Object
Dim libObj As Object
If Not ThisDisplay.InsertLibraryObjectByName("Misc", "Check")
Then
  MsgBox "Error with InsertLibraryObjectByName!"
End If
Set newSym = ThisDisplay.GetHeadObject
MsgBox newSym.GetObjectTypeName

If Not ThisDisplay.ConvertGwxSymbolToLibraryObject(newSym,
libObj) Then
  MsgBox "Error converting symbol to library object!"
Else
  Msgbox "GwxSymbol successfully converted to library
object!"
End If
```

SymbolLibrary Interface

- **AlwaysOnTop As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

MsgBox sl.AlwaysOnTop
sl.AlwaysOnTop = Not sl.AlwaysOnTop
MsgBox sl.AlwaysOnTop
'Should display "False" and then "True"
```

- **Function AddNewCategory(category As String) As Object**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

Dim sc As SymbolCategory
Set sc = sl.AddNewCategory("Test")
If sc Is Nothing Then
  MsgBox "Symbol Category not added!"
  Exit Sub
Else
  Msgbox "Category successfully added!"
End If
```

- **Function RenameCategory(oldCategoryName As String, newCategoryName As String) As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

If Not sl.RenameCategory("Test", "Test1") Then
  MsgBox "Error renaming category! "
  Exit Sub
Else
  Msgbox "Category successfully renamed!"
End If
```

- **Function DeleteCategory(category As String) As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

If Not sl.DeleteCategory("Test1") Then
  MsgBox "Error deleting symbol category!"
```

```
Else
  MsgBox "Category successfully deleted!"
End If
```

- **Function GetNumberOfCategories() As Long**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

MsgBox "Number of categories: " & sl.GetNumberOfCategories
```

- **Function GetCategoryFromIndex(index As Long) As Object**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If


Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromIndex(0)
If sc Is Nothing Then
  MsgBox "Failed to get category from index!"
  Exit Sub
Else
  MsgBox "Category name: " & sc.GetCategoryName
End If
```

- **Function GetCategoryFromName(category As String) As Object**

```
(See InsertLibraryObject)
```

- **Function SetRootDirectory(directory As String) As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

If Not sl.SetRootDirectory("C:\Program Files\ICONICS") Then
  MsgBox "Error setting root directory!"
Else
  MsgBox "Root directory has been set!"
  'Verify that it has actually been set correctly
  'by reopening the Symbol Library
```

```
End If
```

- **Function GetRootDirectory() As String**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

MsgBox sl.GetRootDirectory
```

- **Visible As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

MsgBox sl.Visible
sl.Visible = Not sl.Visible
MsgBox sl.Visible
'Should display "False" and then "True"
```

SymbolCategory Interface

- **EdgeEffect As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Misc")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

MsgBox "Navigate to the Misc category and verify that the"
    & vbCr & "edge effect is opposite " & sc.EdgeEffect & "."
```

```
sl.Visible = True
sl.AlwaysOnTop = True
sc.EdgeEffect = Not sc.EdgeEffect
MsgBox sc.EdgeEffect
```

- **ImageSize As Integer**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Misc")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

sc.ImageSize = BigSize
MsgBox sc.ImageSize        'Should be 80
sc.ImageSize = MediumSize
MsgBox sc.ImageSize        'Should be 60
sc.ImageSize = SmallSize
MsgBox sc.ImageSize        'Should be 40
```

- **Function AddSymbol(symbol As Object) As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Misc")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

Dim symbol As Object
Set symbol = sc.GetSymbolFromName("Check")
If symbol Is Nothing Then
  MsgBox "Failed to get symbol from name!"
```

```
  Exit Sub
End If

Set sc = sl.GetCategoryFromName("Test1")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

If Not sc.AddSymbol(symbol) Then
  MsgBox "Error with AddSymbol!"
Else
  MsgBox "Symbol successfully added!"
End If
```

- **Function RenameSymbol(oldName As String, newName As String) As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If


Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Test1")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

If Not sc.RenameSymbol("Check", "NewCheck") Then
  MsgBox "Error with RenameSymbol!"
Else
  MsgBox "Symbol successfully renamed!"
End If
```

- **Function DeleteSymbol(symbolName As String) As Boolean**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If


Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Test1")
If sc Is Nothing Then
```

```
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

If Not sc.DeleteSymbol("NewCheck") Then
  MsgBox "Error with DeleteSymbol!"
Else
  MsgBox "Symbol successfully deleted!"
End If
```

- **Function GetNumberOfSymbols() As Long**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If


Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Misc")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

MsgBox "Number of symbols in Misc category: " &
sc.GetNumberOfSymbols
```

- **Function GetSymbolFromIndex(index As Long) As Object**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If


Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Misc")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If


Dim symbol As Object
Set symbol = sc.GetSymbolFromIndex(0)
If symbol Is Nothing Then
  MsgBox "Failed to get symbol from name!"
```

```
  Exit Sub
Else
  MsgBox "Successfully retrieved symbol!"
End If
```

- **Function GetSymbolFromName(symbol As String) As Object**

(Tested in InsertLibraryObject)

- **Function GetSymbolName(index As Long) As String**

```
Dim sl As SymbolLibrary.SymbolLibrary
Set sl = New SymbolLibrary.SymbolLibrary
If sl Is Nothing Then
  MsgBox "Symbol Library not instantiated!"
  Exit Sub
End If

Dim sc As SymbolCategory
Set sc = sl.GetCategoryFromName("Misc")
If sc Is Nothing Then
  MsgBox "Failed to get category from name!"
  Exit Sub
End If

Dim symname As String
symname = sc.GetSymbolName(0)
If symname = "" Then
  MsgBox "Symbol doesn't exist or has no name!"
  Exit Sub
Else
  MsgBox "Symbol Name: " & symname
End If
```

- **Function GetCategoryName(index As Long) As String**

```
Dim sl As SymbolLibrary.SymbolLibrary
   Set sl = New SymbolLibrary.SymbolLibrary
   If sl Is Nothing Then
     MsgBox "Symbol Library not instantiated!"
     Exit Sub
   End If

   Dim sc As SymbolCategory
   Set sc = sl.GetCategoryFromIndex(1)
   If sc Is Nothing Then
     MsgBox "Failed to get category from index!"
     Exit Sub
```

```
End If
MsgBox "Category name: " & sc.GetCategoryName
```

Tree Explorer ActiveX OLE Automation Reference

This reference describes the OLE Automation features available in the Tree Explorer ActiveX. The Tree Explorer ActiveX provides a COM interface that allows automation interfaces run from within the ActiveX container to manipulate the Tree Explorer ActiveX control as it is running. The interface is available to all programming languages that support COM, including Visual Basic (VB), Visual Basic for Applications (VBA), and Microsoft Visual C++.

To access the Automation interface from VB and VBA, the Tree Explorer ActiveX must be made available by selecting **TreeExplorerLib** in the Object Browser.

Control Properties

FileName As String

Type: BSTR

Description

Sets/gets the name of the Tree Explorer ActiveX configuration file.

Example

ThisDocument.TreeExplorer1.FileName=”C:\TreeConfig.tre”

ShowNewRecipeDialog As Boolean

Type: Boolean

Description

Enable/disable the message box that ask the user for the new recipe item name when it is created using GraphWorX pick action.

Example

ThisDocument.TreeExplorer1.ShowNewRecipeDialog = True

URLPathName

Type: BSTR

Description

Sets/gets the URL path for a configuration file over the Web. Must set URLPathUsed to “True” for this to take effect.

Example

ThisDocument.TreeExplorer1.URLPathName=
”http://www.iconics.com/treecfg.tre”

URLPathUsed

Type: Boolean

Description

Enables/disables the use of a URL path.

Example

ThisDocument.TreeExplorer1.URLPathUsed=True

Control Methods

AboutBox()

Shows **About** dialog box.

Example

ThisDocument.TreeExplorer1.AboutBox

LoadConfigFile(BSTR newVal) as Boolean

Loads the configuration file specified in newVal.

Example

Dim ret As Boolean

ret = TreeExplorer1.LoadConfigFile("C:\My Configs\cfg.tre")

LoadConfigURL(BSTR newVal) as Boolean

Loads the configuration file specified in the URL string newVal.

Example

Dim ret As Boolean

ret = TreeExplorer1.LoadConfigURL("http://localhost/cfg.tre")

SaveConfigFile(BSTR newVal) as Boolean

Saves your current Tree Explorer ActiveX configuration to the file specified in newVal.

Example

Dim ret As Boolean

ret = TreeExplorer1.SaveConfigFile("C:\My Configs\cfg.tre")

SetRuntimeMode(bEnable As Long)

Forces the Tree Explorer ActiveX to go in runtime/configuration mode.

Example

TreeExplorer1.SetRuntimeMode(True)

Control Events

OnTreeNodeDblClk(inodeID As Integer, strPath As String, bExecute As Integer)

Description

The Tree Explorer ActiveX will fire up this event when the user double-clicks on any node of the tree.

It will return:

inodeID containing the unique identification of the tree node.

strPath containing the path from the root up to the clicked node.

bExecute is used to validate the execution of the file if the tree is a file explorer tree. If the user changes this value from 1 to 0 the file explorer will not execute the associated file if the node represents a valid file.

TrendWorX32 OLE Automation Reference

TrendWorX32 Container exposes a rich and useful set of OLE Automation properties and methods. The various OLE objects exposed are:

- **Application Object (Twx32App)**
- **Documents Object (Twx32Documents)**
- **Document**

Application Object (Twx32App)

The **Application Object** has the following properties and methods:

Properties

- **BSTR Name**
- **boolean Visible**
- **boolean RuntimeMode**
- **IDispatch* Application**
- **IDispatch* Documents**

Methods

- **boolean EnterRuntime()**
- **boolean ExitRuntime()**
- **void Quit()**
- **boolean LoadBgTrend()**
- **boolean ShutDownBgTrend()**
- **void CloseAllDocuments()**

Documents Object (Twx32Documents)

The **Documents** property exposes another OLE object, which can be used to manipulate individual TrendWorX32 displays. It has the following properties and methods:

Properties

- **long Count**

Methods

- **boolean Open(BSTR FilePath)**
- **boolean Add(BSTR Caption)**
- **boolean RunDocument(BSTR Item)**
- **boolean CfgDocument(BSTR Item)**
- **boolean PrintDocument(BSTR Item)**
- **boolean IsDocRunning(BSTR Item)**
- **boolean CloseDocument(BSTR Item)**
- **boolean CloseSaveDocument(BSTR Item)**

Document

TrendWorX32 Container supports the following properties, methods, and events at the **Document** (Display) level:

Properties

- **IDispatch* Application**
- **BSTR FullName**
- **BSTR Path**
- **boolean Saved**
- **boolean RuntimeMode**
- **boolean VBADesignMode**
- **boolean Maximized**
- **BSTR Caption**
- **boolean Visible**
- **boolean IgnoreBackColor**
- **boolean IgnoreObjectBackColor**
- **boolean GridsOn**

Methods

- **void EnterRuntime()**
- **void ExitRuntime()**
- **long NumberOfObjects()**
- **void Activate()**

- **void Save()**
- **void SaveAs(BSTR fileName)**
- **void Print() ***

*** Note:** For the document print method, use the following code in a VBA application:

```
Private Sub CommandButton1_Click()
Dim Twx32App As Twx32.Application
Dim TwX32Documents As Twx32.Documents
Dim strName As String
' Get Doc Name
strName = This Document.Caption
' Get Application
Set Twx32App = ThisDocument.Application
' Get Documents Collection
Set Twx32Documents = Twx32App.Documents
' Print
Twx32Documents.PrintDocument strName
```

Events

- **void Open()**
- **void Close()**
- **void Runtime()**
- **void Configure()**
- **void NewTrend()**
- **void NoSecurity()**
- **void NoKey()**
- **void Print()**

OLE Automation Examples and Samples

TrendWorX32 includes a set of VBA examples under the "\ICONICS\GENESIS32\Examples\TrendWorX32 Examples" directory. The following folders located inside the "Examples" directory are associated with the Viewer.

Aliasing. The files in this folder demonstrate the use of the OLE Automation interface of the TrendWorX32 Viewer ActiveX within a VBA-enabled application, such as GraphWorX32, in aliasing.

Excel Templates. The files in this folder are example Excel templates that can be used when creating Excel workbooks to be used with TrendWorX32 Reporting.

Real-Time Trend Configuration. The files in this folder demonstrate the use of the OLE Automation interface of the TrendWorX32 Viewer ActiveX within a VBA-enabled application, such as GraphWorX32. It demonstrates how to add pens "on the fly" by reading the pen configuration from a Microsoft Access database (.mdb) file and utilizing the OLE Automation properties of the TrendWorX32 ActiveX through VBA scripts.

TrendWorX32 Data Logging and Reporting. This folder contains sample TrendWorX32 data-logging configuration, a sample historical database, as well as sample reporting and historical replay displays.

TrendWorX32 Backgrounds. This folder contains several image files that can be used as backgrounds for the TrendWorX Viewer ActiveX display.

TrendWorX32 VBA Examples. This folder contains various examples that demonstrate the use of the OLE Automation interface of the various TrendWorX32 modules and components.

VB Script Examples. This folder contains a **TWX-VBScriptEx.gdf** display that uses VB scripting.

TrendWorX32 Persistent Trending OLE Automation Reference

The Persistent Trending application exposes, through OLE automation, its document as a single-instance shared object. The object's name is **TWXBgTrend.Document,** and it is registered in the system registry. Once launched, the Persistent Trending server will register its document as the Active Object with the OLE Running Object Table (ROT), and therefore all client applications can access this instance rather than create a new instance each time they desire to connect to the Persistent Trending server. In Visual Basic code, this is equivalent to:

```
Dim myTWXServer As Object

‘ get running object
Set myTWXServer = GetObject(, ’’TWXBgTrend.Document’’)
```

Every client application should always try to get the running instance of the server rather than create a new instance (except for the first time connection). The **TWXBgTrend.Document** OLE automation object supports the following OLE automation properties and methods:

OLE Automation Properties

BOOL Visible	TRUE if the application window is visible; supports SET/GET
BSTR FullName	Returns the Full path and file name of the currently loaded file in TWXBgTrend; supports GET only
BOOL Maximized	TRUE if the application is maximized; supports GET/SET
BSTR Name	Returns the file name of the current file loaded in Persistent Trending; supports GET only
BSTR Path	Returns the path for the currently loaded Persistent Trending file; supports GET only
BOOL ReadOnly	Returns TRUE if the application does not allow any modifications; supports GET only
BOOL Runtime	Returns TRUE if the application is in runtime mode; supports GET only
BOOL Saved	Returns TRUE if all pending changes are saved or no new changes have been made; supports GET only
long GroupCount	Returns the number of groups configured in Persistent Trending; supports GET only

OLE Automation Methods

1. **void Close(BOOL SaveChanges, BSTR File)**

   Call this method to shut down the application. If "SaveChanges" is TRUE, the last changes made will be saved to the user-supplied file.

2. **void Open(BSTR File)**

   Call this method to open the user-supplied File in Persistent Trending.

3. **void SaveAs(BSTR File)**

   Call this method to save the current Persistent Trending configuration file in "File."

4. **void EnterRuntime()**

   Call this method to commit the Persistent Trending server into runtime mode. Data collection will start (if desired) automatically for the group, and data logging (if configured) will also be enabled.

5. **void ExitRuntime()**

   Call this method to exit runtime mode.

6. **void ActivateGroup(BSTR GroupName)**

   Call this method to activate a specific group (i.e. start data collection and data logging if configured). Note that each group configured in the Persistent Trending server can start data collection automatically upon entering runtime mode or when you desire by using this OLE automation method.

7. **void DeActivateGroup(BSTR GroupName)**

   Call this method to deactivate a group. This will stop data collection and buffering in memory. If data logging is enabled for this group, all collected data up to the time of application of this method will be logged to disk.

8. **long GetGroupRate(BSTR GroupName)**

   Call this method to obtain the data-collection rate for a specific group.

9. **long GetGroupSamples(BSTR GroupName)**

   Call this method to obtain the number of samples the group is currently buffering in memory.

10. **BOOL GetGroupActivate(BSTR GroupName)**

    Call this method to find out if the specific group has been configured to activate data collection and buffering upon Persistent Trending entering runtime mode.

11. **long GetGroupPointCount(BSTR GroupName)**

Call this method to obtain the number of signals configured for the specific group.

12. BOOL GetPointData(BSTR PointName, long Rate, long Samples, BOOL Stamp, VARIANT* Values, VARIANT* TDates, VARIANT* MSecs, VARIANT* Quality)

Call this method to retrieve buffered data in memory from Persistent Trending for a specific signal ("PointName"). "Rate" is the desired time resolution of the returned samples. "Samples" is the desired number of samples to be returned. The variants "Values," "TDates," "MSecs," and "Quality" represent variant arrays (of element types VARIANT), which contain the values, time and date stamps, millisecond resolution, and quality information about the retrieved samples.

"Values" variants utilize double values (VT_R8), and "TDates" variants utilize "DATE" types (VT_DATE). "Msecs" and "Quality" variants utilize VT_I4 types.

The Persistent Trending will allocate memory for these arrays, and the caller of this method must always FREE the associated memory. Also, if no samples are retrieved, the method will return FALSE and the variants will be empty.

Upon executing this method, the Persistent Trending server will search *all* groups configured, for the first group, which has the "PointName" signal configured. Once found, it will perform a time-stamped search to retrieve all values buffered in memory that satisfy the desired data-collection rate and number of returned samples, starting from the most current one. Persistent Trending will search for the requested data at time intervals reflecting the desired data resolution. The last valid sample within each search interval will be returned to the caller. If "bStamp" is set to TRUE, the actual time and date stamp of the retrieved sample will be modified to reflect the time and date stamp of the search interval.

13. BOOL GetGroups(VARIANT* GroupNames)

Call this method to obtain all the group names configured in the Persistent Trending server. They are returned in the array of BSTR's "GroupNames," as before Persistent Trending will allocate memory for these arrays, the caller of this method must always FREE the associated memory. If no groups are configured, the method returns FALSE.

14. BOOL GetGroupPoints(BSTR GroupName, VARIANT* GroupPoints)

Call this method to retrieve the signal names configured for a given group. They are returned in the array of BSTR's "GroupPoints," and before Persistent Trending will allocate memory for these arrays, the caller of this method must always FREE the associated memory. If no signals are configured, the method returns FALSE.

15. BOOL GetGroupLogging(BSTR GroupName)

Call this method to find whether data logging has been enabled for a specific group. The method returns TRUE if data logging is enabled.

16. **BSTR GetGroupFile(BSTR GroupName)**

    Call this method to retrieve the current "FileName" to which a specific group is data logging.

17. **BOOL GetGroupPointData(BSTR GroupName, BSTR PointName, long Rate, long Samples, BOOL Stamp, VARIANT* Values, VARIANT* TDates, VARIANT* MSecs, VARIANT* Quality)**

    Call this method to retrieve buffered data in memory from Persistent Trending for a specific signal ("PointName"). "Rate" is the desired time resolution of the returned samples. "Samples" is the desired number of samples to be returned. The variants "Values," "TDates," "MSecs," and "Quality" represent variant arrays (of element types "VARIANT"), which contain the values, time and date stamps, millisecond resolution, and quality information about the retrieved samples. "Values" variants utilize double values (VT_R8), "Tdates" variants utilize "DATE" types (VT_DATE), and "Msecs" and "Quality" utilize VT_I4 types.

    Persistent Trending will allocate memory for these arrays, and the caller of this method must always FREE the associated memory. Also, if no samples are retrieved, the method will return FALSE and the variants will be empty.

    Upon executing this method, the Persistent Trending server will search all Persistent Trend groups for a group called "GroupName," and then it will search for the "PointName" signal in the requested group. Then it will perform a time-stamped search to retrieve all values buffered in memory that satisfy the desired data-collection rate and the number of returned samples, starting from the most current one. Persistent Trending will search for the requested data at time intervals reflecting the desired data resolution. The last valid sample within each search interval will be returned to the caller. If "bStamp" is set to TRUE, the actual time and date stamp of the retrieved sample will be modified to reflect the time and date stamp of the search interval.

18. **BOOL GetPointStat(BSTR GroupName, BSTR PointName, VARIANT* StatValues)**

    Call this method to retrieve statistical information about a specific signal belonging to a specific group. The returned array of variants "StatValues" will have the following as "VARIANT" elements:

    - Min value
    - Max value
    - Average value
    - Standard deviation
    - Starting time for statistics
    - Ending time for statistics

    Persistent Trending will allocate memory for these arrays, and the caller of this method must always FREE the associated memory. Also, if no samples are retrieved, the method will return FALSE and the variants will be empty.

19. **BOOL GetGroupPointAt(BSTR GroupName, BSTR PointName, DATE TDateAt, VARIANT* Value, VARIANT* TDateFound, VARIANT* Qual)**

    Call this method to retrieve a sample for a given time and date from Persistent Trending. All the returned information is in variants, and if the method succeeds, it returns TRUE.

20. **BOOL GetGroupPointDataAt(BSTR GroupName, BSTR PointName, long Rate, long Samples, BOOL Stamp, DATE TDateFrom, VARIANT* Values,VARIANT* TDates, VARIANT* MSecs, VARIANT* Quality)**

    This method is the same as "GetGroupPointData," with the exception that now you can specify a starting time and date stamp in "TDateFrom" to retrieve data from Persistent Trending's memory buffers.

21. **BOOL GetPointDataAt(BSTR PointName, long Rate, long Samples, boolean Stamp, DATE TDateFrom, VARIANT* Values, VARIANT* TDates, VARIANT* MSecs, VARIANT* Quality)**

    This method is the same as "GetPointData," with the exception that now you can specify a starting time and date stamp in "TDateFrom" to retrieve data from Persistent Trending's memory buffers.

22. **void Refresh()**

    Call this method to refresh the **Persistent Trending** window. This method should be called only once.

23. **boolean GetPointDataUTC(BSTR GroupName, BSTR PointName, long Rate, long Samples, BOOL Stamp, VARIANT* Values, VARIANT* TDates, VARIANT* MSecs, VARIANT* Quality)**

    Call this method to retrieve buffered data in memory from Persistent Trending for a specific signal ("PointName"). "Rate" is the desired time resolution of the returned samples. "Samples" is the desired number of samples to be returned. The variants "Values," "TDates," "MSecs," and "Quality" represent variant arrays (of element types "VARIANT"), which contain the values, time and date stamps, millisecond resolution, and quality information about the retrieved samples. "Values" variants utilize double values (VT_R8), "Tdates" variants utilize "DATE" types (VT_DATE), and "Msecs" and "Quality" utilize VT_I4 types.

    Persistent Trending will allocate memory for these arrays, and the caller of this method must always FREE the associated memory. Also, if no samples are retrieved, the method will return FALSE and the variants will be empty.

    Upon executing this method, the Persistent Trending server will search all Persistent Trend groups for a group called "GroupName," and then it will search for the "PointName" signal in the requested group. Then it will perform a time-stamped search to retrieve all values buffered in memory that satisfy the desired data-collection rate and the number of returned samples, starting from the most current one. Persistent Trending will search for the requested data at time intervals reflecting the desired data resolution. The last valid sample within each search interval will be returned to the caller. If "bStamp" is set to TRUE, the actual time and date stamp of the retrieved

sample will be modified to reflect the time and date stamp of the search interval. Time stamps are returned in UTC time format.

24. **boolean GetGroupPointDataUTC(BSTR GroupName, BSTR PointName, long Rate, long Samples, BOOL Stamp, VARIANT* Values, VARIANT* TDates, VARIANT* MSecs, VARIANT* Quality);**

    Call this method to retrieve buffered data in memory from Persistent Trending for a specific signal ("PointName"). "Rate" is the desired time resolution of the returned samples. "Samples" is the desired number of samples to be returned. The variants "Values," "TDates," "MSecs," and "Quality" represent variant arrays (of element types "VARIANT"), which contain the values, time and date stamps, millisecond resolution, and quality information about the retrieved samples. "Values" variants utilize double values (VT_R8), "Tdates" variants utilize "DATE" types (VT_DATE), and "Msecs" and "Quality" utilize VT_I4 types.

    Persistent Trending will allocate memory for these arrays, and the caller of this method must always FREE the associated memory. Also, if no samples are retrieved, the method will return FALSE and the variants will be empty.

    Upon executing this method, the Persistent Trending server will search all Persistent Trend groups for a group called "GroupName," and then it will search for the "PointName" signal in the requested group. Then it will perform a time-stamped search to retrieve all values buffered in memory that satisfy the desired data-collection rate and the number of returned samples, starting from the most current one. Persistent Trending will search for the requested data at time intervals reflecting the desired data resolution. The last valid sample within each search interval will be returned to the caller. If "bStamp" is set to TRUE, the actual time and date stamp of the retrieved sample will be modified to reflect the time and date stamp of the search interval. Time stamps are returned in UTC time format.

25. **boolean GetPointStatUTC(BSTR GroupName, BSTR PointName, VARIANT* StatValues);**

    Call this method to retrieve statistical information about a specific signal belonging to a specific group. The returned array of variants "StatValues" will have the following as "VARIANT" elements:

    - Min value
    - Max value
    - Average value
    - Standard deviation
    - Starting time for statistics
    - Ending time for statistics

    Persistent Trending will allocate memory for these arrays, and the caller of this method must always FREE the associated memory. Also, if no samples are retrieved, the method will return FALSE and the variants will be empty. Time stamps are returned in UTC time format.

26. **boolean GetGroupPointAtUTC(BSTR GroupName, BSTR PointName, DATE TDateAt, VARIANT* Value, VARIANT* TDateFound, VARIANT* Qual)**

    Call this method to retrieve a sample for a given time and date from Persistent Trending. All the returned information is in variants, and if the method succeeds, it returns TRUE. Time stamps are returned in UTC time format.

27. **boolean GetGroupPointDataAtUTC(BSTR GroupName, BSTR PointName, long Rate, long Samples, boolean Stamp, DATE TDateFrom, VARIANT* Values, VARIANT* TDates, VARIANT* MSecs, VARIANT* Quality)**

    This method is the same as "GetGroupPointData," with the exception that now you can specify a starting time and date stamp in "TDateFrom" to retrieve data from Persistent Trending's memory buffers. Time stamps are returned in UTC time format.

28. **boolean GetPointDataAtUTC(BSTR PointName, long Rate, long Samples, boolean Stamp, DATE TDateFrom, VARIANT* Values, VARIANT* TDates, VARIANT* MSecs, VARIANT* Quality)**

    Call this method to retrieve buffered data in memory from Persistent Trending for a specific signal ("PointName"). "Rate" is the desired time resolution of the returned samples. "Samples" is the desired number of samples to be returned. The variants "Values," "TDates," "MSecs," and "Quality" represent variant arrays (of element types VARIANT), which contain the values, time and date stamps, millisecond resolution, and quality information about the retrieved samples.

    "Values" variants utilize double values (VT_R8), and "TDates" variants utilize "DATE" types (VT_DATE). "Msecs" and "Quality" variants utilize VT_I4 types.

    The Persistent Trending will allocate memory for these arrays, and the caller of this method must always FREE the associated memory. Also, if no samples are retrieved, the method will return FALSE and the variants will be empty.

    Upon executing this method, the Persistent Trending server will search *all* groups configured, for the first group, which has the "PointName" signal configured. Once found, it will perform a time-stamped search to retrieve all values buffered in memory that satisfy the desired data-collection rate and number of returned samples, starting from the most current one. Persistent Trending will search for the requested data at time intervals reflecting the desired data resolution. The last valid sample within each search interval will be returned to the caller. If "bStamp" is set to TRUE, the actual time and date stamp of the retrieved sample will be modified to reflect the time and date stamp of the search interval. Time stamps are returned in UTC time format.

29. **boolean GetGroupPointInfo(BSTR GroupName, BSTR PointName, VARIANT* HighRange, VARIANT* LowRange, VARIANT* EngUnits, VARIANT* Description);**

This method returns point information for a given signal, such as "HighRange" and "LowRange," "EngUnits," and "Description," given a Persistent Trend "GroupName."

30. **boolean GetPointInfo(BSTR PointName, VARIANT* HighRange, VARIANT* LowRange, VARIANT* Description, VARIANT* EngUnits);**

    This method returns point information for the first signal, such as "HighRange" and "LowRange," "EngUnits," and "Description," that matches "PointName."

ProjectWorX32 Support

Several OLE automation methods facilitate ProjectWorX32 deployment and WebHMI enhanced support:

1. **ReplaceStringInString(**
   StringToReplace as String,
   OldSubstring as String,
   NewSubstring as String
) as String

2. **ReplaceHost(**
   OldHostName as String,
   NewHostName as String
) as Long

3. **ReplaceFilePath(**
   OldSubstring as String,
   NewSubstring as String
) as Long

4. **ReplaceTag(**
   OldSubstring as String,
   NewSubstring as String
) as Long

5. **ReplaceHostEx(**
   OldHostNameSubstring as String,
   NewHostNameSubstring as String
   MatchCase as Boolean,
   MatchWholeWord as Boolean
) as Long

The **ReplaceHostEx** with parameters **MatchCase=False & MatchWholeWord=True** works exactly the same as the **ReplaceHost()** function.

Using the Persistent Trending OLE Automation Server

The purpose of the Persistent Trending OLE automation server is to function primarily as a memory-based data-logging engine, which buffers data for user-configured signals at desired data-collection rates in memory. In addition, the buffered data can be configured to be data-logged in a .csv or .txt file.

The Persistent Trending automation server is a stand-alone (EXE) server, which registers as the active running object with the tables maintained by OLE. This means that there is only one instance of Persistent Trending running per computer, and all interested clients should get an interface pointer to that running instance (rather than create a new Persistent Trending object, which will fail if Persistent Trending is already running).

The Persistent Trending server should be started and running at the time of committing a process control/HMI application to runtime mode, so it will have enough time to buffer data before any client requests are processed. There are several ways this can be done.

You can start the Persistent Trending server manually, from the Program Group, or place it in the StartUp Folder. In either case, the Persistent Trending server will load the most-recent configured file. Then it can be committed into runtime mode by choosing **Start Logging** from the **Actions** menu, or clicking the **Traffic Light** button on the toolbar.

You can evoke the Persistent Trending server programmatically. The following is a small Visual Basic code sample used to initiate the Persistent Trending server, to load a desired configuration file, and to enter runtime mode:

```
Dim myTrend As Object

 ‘ Create Persistent Trending Server Object

Set myTrend = CreateObject("TWXBgTrend. Document")

‘ load configuration file

myTrend.Open "D:\MyDirFE32\sim10.b32"

‘ Show Persistent Trending Window

myTrend.Visible = True
‘ Refresh Persistent Trending Window

myTrend.Refresh

‘ Enter Runtime
myTrend.EnterRuntime
```

Once this code is executed, Persistent Trending will initiate, load the desired configuration file, show its window, and enter runtime mode. At this point, any automation client of the Persistent Trending server should be able to get an instance to the running TWXBgTrend object, and it will be able to interact with the server completely.

Here is another Visual Basic code sample used to actually retrieve data from Persistent Trending's buffers:

```
Dim myTrendObject As Object

‘ Get Running Object

Set myTrendObject = GetObject(, "TWXBgTrend.Document")

‘ string sPoint is the desired Point Name for which data will be ‘
retrieved
' Set up variables to receive data from Background Trending Server
 ‘ A variant array for sample values

Dim Values As Variant

‘ A variant array for Time and Date stamps

Dim TDates As Variant

‘ A variant array for Msecs timestamps

Dim MSecs As Variant

‘ A variant array for OPC quality flags
Dim Qual As Variant

‘ Index used for iteration

Dim I As Long

 ‘ RateVal is a long representing the desired data resolution rate

‘ SamplesVal is a long representing the desired number of
‘ samples to retrieve

Dim sPointVal As String
Dim vEntry As Variant
```

```
Dim bRet As Boolean
Dim strValue As String

' Call Background Trending automation method

bRet = myTrendObject.GetPointData(sPoint, RateVal, SamplesVal,
FALSE, Values, TDates, MSecs, Qual)

' Fill In Values in a listbox control called PointValues

PointValues.Clear

'Fill In !!!

For I = LBound(Values) To UBound(Values)
    vEntry = Values(I)
    strValue = TypeName(vEntry)
    sPointVal = Str(vEntry)
    PointValues.AddItem sPointVal
Next I
```

Note that error-checking code is omitted for clarity. The caller should always check for empty arrays, in case there were no data to return.

Persistent Trending OLE Automation Examples

The GENESIS32 installation contains a sample TrendWorX32 file, called **PersistentTrendVBADemo.t32,** that demonstrates how to use Persistent Trending's OLE automation properties. There is also a **readme.txt** file that explains how to use this sample Visual Basic application.

TrendWorX32 Reporting OLE Automation Reference

The TrendWorX32 Reporting Application registers with the OLE ROT (Running Object Table) as a single-instance OLE Automation server. If it has started, all clients can connect to it by using a "GetObject" call (i.e. connecting to the running instance of reporting). The TrendWorX32 Reporting OLE Automation server currently supports the following OLE Automation properties and methods.

OLE Automation Properties

boolean Visible	Shows/hides application window.
boolean Maximized	Maximizes/minimizes application window.

OLE Automation Methods

void Refresh()	Redraws the view.
void Open(BSTR FileName)	Opens a configuration file.
void StartReports()	Starts runtime mode.
void StopReports()	Exits runtime mode.
booleanAreReports Running()	Returns true if in runtime mode.
voidClose(boolean SaveChanges, BSTR file)	Exits the report.

A set of OLE automation methods has been added to allow for flexible report manipulation. For these OLE automation methods to be used effectively, you should call first the **GetReportNames** method to get a listing of all the currently available reports and then use automation to set the desired time span, etc. Depending on the status of each report or the periodic schedule, a VBA-enabled application can filter which reports will be available to the operator for manipulation. Note that the best way to make use of operator-based triggering of report execution within a VBA-enabled application, such as GraphWorX32, is to configure immediate report types, and do not enable the desired reports. Then, the operator through the newly added OLE Automation support will be able to start or stop existing reports.

These methods are:

VARIANT GetReportNames()	Returns a Safearray with all the report names.
short GetReportStatus(BSTR ReportName)	Returns an enumerated type TWXREPORTSTATUS with the report status for each report by name, which can be of the following enumerated values: RptStatusOk = 0 RptStatusEditing = 1 RptStatusDeleting = 2 RptStatusExecuting = 3 RptStatusExiting = 4 RptStatusPaused = 5 RptStatusCancelled = 6 RptStatusStandby = 7 RptStatusNotFound = 8 RptStatusConfigErrors = 9 RptStatusConfigure = 10
short GetReportScheduleType(BSTR ReportName)	Returns an enumerated type TWXREPORTSCHEDULE with the desired report schedule type: RptNotFound = 0 RptImmediate = 1 RptOnce = 2 RptPeriodic = 3 RptEvents = 4
boolean SetReportTimeSpan(BSTR ReportName, DATE StartDateTime, DATE EndDateTime)	Sets the report time range for immediate type of reports which currently are not executing.
boolean SetReportTarget(BSTR ReportName, BSTR ReportTarget)	Sets the report target name for the desired report. The report has to be disabled first using the DisableReport method if necessary.
short RunReport(BSTR ReportName)	Executes the desired report returning a TWXREPORTSTATUS status enumeration with the status of the report. Works only in runtime mode.
short StopReport(BSTR ReportName)	Stops execution of the desired report. Works only in runtime mode. It returns a

	TWXREPORTSTATUS status enumeration with the status of the report.
short AddNewReport(BSTR ReportName, BSTR ReportTargetName, long ReportDataFilter, long ReportFilterOrder, boolean Qualities, boolean CopyOver, long ProcessDataSecs, long ProcessDataMSecs)	Adds a new report with basic configuration settings. The report is initially disabled. It returns a TWXREPORTEDIT edit status code. The ReportDataFilter is of type GFWREPORT_DATA_FILTERS. The ReportFilterOrder parameter is used only in Moving filters, and the ProcessDataSecs, ProcessDataMSecs define the report processing interval for non-Raw reports
short SetReportCSVTarget(BSTR ReportName, BSTR ReportCSVPath, BSTR ReportCSVNumberFormat, boolean AddHeader, boolean CreateNewWithDate)	Sets the target of an existing report to be a .csv file. The report needs to be disabled first, and the ReportCSVPath refers only to the file path. It returns a TWXREPORTEDIT edit status code.
short SetReportEXCELTarget(BSTR ReportName, BSTR ReportXLSTemplate, BSTR ReportXLSPath, boolean PublishHTML, BSTR ReportHTMLPath, boolean UseXLSWKSheet, BSTR ReportWKSheetName, long ReportXLSRow, long ReportXLSColumn, boolean ReportAutoprint, boolean AddHeader, boolean ReportAutoFormat, boolean ReportXLSProtect, boolean ReportXLSEmail, BSTR ReportXLSEmailAddress, BSTR ReportXLSEmailSubject)	Sets the target of an existing report to be an Microsoft Excel workbook. The report needs to be disabled first, and the ReportXLSPath refers only to the file path. It returns a TWXREPORTEDIT edit status code.
short SetReportDBTarget(BSTR ReportName, BSTR DBConnString, long DBType, boolean CreateNewWithDate)	Sets the target of an existing report to be a database table. The report needs to be disabled first, and the DBConnString refers to an ADO ODBC connection string. It returns a TWXREPORTEDIT edit status code. The DBTYpe is of

	TWXREPORTSDBTYPE enumeration.
short SetReportImmediate(BSTR ReportName, DATE StartTime, DATE EndTime)	Sets the schedule of an existing report to be of Immediate type. The report needs to be disabled. It returns a TWXREPORTEDIT edit status code.
short SetReportOnce(BSTR ReportName, DATE DateToRun, DATE StartTime, DATE EndTime)	Sets the schedule of an existing report to be of Single execution type (Once). The report needs to be disabled. It returns a TWXREPORTEDIT edit status code.
short SetReportEvent(BSTR ReportName, BSTR EventCondition, long ReportRangeSecs, boolean StartFromLastTime)	Sets the schedule of an existing report to be of Event execution type. The report needs to be disabled. It returns a TWXREPORTEDIT edit status code. The EventCondition represents an OPC tag-based expression.
short SetReportHourly(BSTR ReportName, DATE TimeToRun, long Hours, long Minutes, long OffSetMinutes, boolean UseReportTimeSpan, long ReportTimeSpanSecs)	Sets the schedule of an existing report to be of periodic (Hourly) execution type. The report needs to be disabled. It returns a TWXREPORTEDIT edit status code. The UseReportTimeSpan enables the time range of the report not to be defined by the periodic interval but to be set by the user. The TimeToRun is the desired time of the first report execution.
short SetReportDaily(BSTR ReportName, DATE TimeToRun, long OffSetMinutes, boolean UseReportTimeSpan, long ReportTimeSpanSecs)	Sets the schedule of an existing report to be of periodic (Daily) execution type. The report needs to be disabled. It returns a TWXREPORTEDIT edit status code. The UseReportTimeSpan enables the time range of the report not to be defined by the periodic interval but to be set by the user. The TimeToRun is the desired time of the first report execution.
short SetReportWeekly(BSTR ReportName, DATE TimeToRun, long DayOfTheWeek, long OffSetMinutes, boolean UseReportTimeSpan, long ReportTimeSpanSecs)	Sets the schedule of an existing report to be of periodic (Weekly) execution type. The report needs to be disabled. It returns a TWXREPORTEDIT edit status code. The UseReportTimeSpan enables the time range of the report not to be defined by the periodic interval but to be set by the user. The TimeToRun is the desired time of the first report execution. The DayOfTheWeek refers to the day of the week for the report to execute (with Sunday = 1 and Saturday =

	7).
short SetReportMonthly(BSTR ReportName, DATE TimeToRun, boolean FirstDayOfWeek, long OffSetMinutes, boolean UseReportTimeSpan, long ReportTimeSpanSecs)	Sets the schedule of an existing report to be of periodic (Weekly) execution type. The report needs to be disabled. It returns a TWXREPORTEDIT edit status code. The UseReportTimeSpan enables the time range of the report not to be defined by the periodic interval but to be set by the user. The TimeToRun is the desired time of the first report execution. The DayOfTheWeek refers to the day of the week for the report to execute (with Sunday = 1 and Saturday = 7).
short EnableReport(BSTR ReportName)	Enables a given report for execution.
short DisableReport(BSTR ReportName)	Disables a given report for execution.
short AddReportTag(BSTR ReportName, BSTR TagLoggingName, BSTR TagDescrText, BSTR TagEngUnits, BSTR TagDBConnString, long DBType , BSTR TagDBaseGroupName, BSTR TagLogGroupName, long TagDBHandle)	Adds a a new report tag to an existing report. The report needs to be disabled. It returns a TWXREPORTEDIT edit status code. The TagLoggingName is the one defined in the TrendWorX32 Configurator, while the TagDBConnString parameter refers to the ADO ODBC connection string for the historical database. The TagDBHandle parameter is the index of the specific tag in the _Tags table of the historical database.
short DeleteReport(BSTR ReportName)	This method deletes a report. The report needs to be disabled. It returns a TWXREPORTEDIT edit status code.
short SetReportTotalizerParams(BSTR ReportName, long TotalizerUnits, long TotalizerType, long TotalizerDirection, double TotalizerValue, double TotalizerAutoResetLevel)	This method sets the Totalizer filter related parameters. The TotalizerUnits can be any of: Seconds = 0 Minutes = 1 Hours = 2 Days = 3

	The TotalizerType can be of type: Continuous = 0 PreLoad = 1 Preset = 2 The TotalizerDirection can be 0, for counting upwards, or 1 for counting downwards The Totalizer value is simply the Preload or Preset value depending on the Totalizer type.

For these OLE automation methods to be used effectively, you should call first the **GetReportNames** method to get a listing of all the currently available reports and then use automation to set the desired time span, etc. Depending on the status of each report or the periodic schedule, a VBA-enabled application can filter which reports will be available to the operator for manipulation. Note that the best way to make use of operator-based triggering of report execution within a VBA-enabled application, such as GraphWorX32, is to configure immediate report types, and do not enable the desired reports. Then, the operator through the newly added OLE Automation support will be able to start or stop existing reports.

Example

The sample code below adds a simple MS EXCEL Report:

```
Private Sub cmdReports_Click()

Dim reports As TWXReport.Document

Set reports = GetObject(, "TWXReport.Document")


'Add a report

reports.AddNewReport TextBox1.Text, TextBox1.Text, TWXReport.Raw, 8,
False, False, 60, 0

reports.SetReportImmediate TextBox1.Text, Now, Now - 1

reports.SetReportEXCELTarget TextBox1.Text, "D:\Logging Tests\TWX
EXCEL Templates\Production -- Daily Yield.xlt", "D:\Logging Tests\Test
Reports", False, "", False, "", 3, 3, False, True, True, False, False, "", ""


'Add a tag

reports.AddReportTag TextBox1.Text, "SimulatePLC.Random",
"SimulatePLC.Random", "", "Provider=MSDASQL.1;Extended
Properties=""DSN=TWXREPORT_MSJET;DBQ=D:\Logging Tests\Logging
Data\TWXREPORT_MSJET.mdb;DriverId=25;FIL=MS
```

Access;MaxBufferSize=2048;PageTimeout=5;UID=admin;""", 1, "MSJET", "JetGroup", 3

'Execute Report

reports.StartReports

reports.EnableReport TextBox1.Text

reports.RunReport TextBox1.Text

ProjectWorX32 Support

Several OLE Automation methods have been added to facilitate ProjectWorX32 deployment and WebHMI enhanced support:

1. **ReplaceStringInString(**
   StringToReplace as String,
   OldSubstring as String,
   NewSubstring as String
) as String

2. **ReplaceHost(**
   OldHostName as String,
   NewHostName as String
) as Long

3. **ReplaceFilePath(**
   OldSubstring as String,
   NewSubstring as String
) as Long

4. **ReplaceTag(**
   OldSubstring as String,
   NewSubstring as String
) as Long

5. **ReplaceHostEx(**
   OldHostNameSubstring as String,
   NewHostNameSubstring as String
   MatchCase as Boolean,
   MatchWholeWord as Boolean
) as Long

The **ReplaceHostEx** with parameters **MatchCase=False & MatchWholeWord=True** works exactly the same as the **ReplaceHost()** function.

The following is a simple example of using the OLE automation methods of TrendWorX32 Reporting added for Version 7.x. It assumes that the TrendWorX32 Reporting application is already running. If not, a Shell command can be used to start it.

```
Private Sub cmdReports_Click()

Dim reports As TWXReport.Document

Set reports = GetObject(, "TWXReport.Document")


'Add a report

reports.AddNewReport “MyReport”, “MyReport”, TWXReport.Raw, 8, False, False, 60, 0

reports.SetReportImmediate “MyReport”, Now, Now - 1

'reports.SetReportEXCELTarget “MyReport”, "D:\Logging Tests\TWX EXCEL Templates\Production -- Daily Yield.xlt", "D:\Logging Tests\Test Reports", False, "", False, "", 3, 3, False, True, True, False, False, "", ""

reports.SetReportCSVTarget “MyReport”, "D:\Logging Tests", "x.xxxxx", False, False


'Add tags

reports.AddReportTag “MyReport”, "SimulatePLC.Random", "SimulatePLC.Random", "", "Provider=MSDASQL.1;Extended Properties=""DSN=TWXREPORT_MSJET;DBQ=D:\Logging Tests\Logging Data\TWXREPORT_MSJET.mdb;DriverId=25;FIL=MS Access;MaxBufferSize=2048;PageTimeout=5;UID=admin;""", 1, "MSJET", "JetGroup", 3


' Get Report Names

Dim ReportNames As Variant

ListBox1.Clear

ReportNames = reports.GetReportNames

For i = LBound(ReportNames) To UBound(ReportNames)
```

```
    ListBox1.AddItem ReportNames(i)
Next i

'Execute Report
reports.StartReports
reports.EnableReport “MyReport”
reports.RunReport “MyReport”

'Delete Report
'reports.DeleteReport MyReport

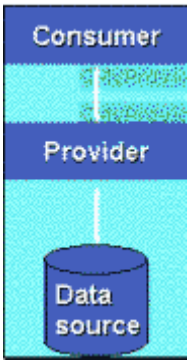
End Sub
```

TrendWorX32 OLE DB Provider

OLE DB is the dominant data access methodology from Microsoft that allows high-performance data access from any data source. Any tabular data is viewable through OLE DB regardless of whether it came from a database. This flexibility gives developers a tremendous amount of power.

In version 7.x, the TrendWorX32 OLE DB Provider included the following new features:

- New database connection dialog
- Enhanced data retrieval support



OLE DB Architecture

The design of OLE DB includes the concept of consumer and provider. The figure above shows a graphical representation of the OLE DB system. The consumer represents the traditional client. The provider places data into a tabular format and returns it to the client. A provider is a set of COM components that contains a series of interfaces. Because these are standard interfaces, any OLE DB consumer can access data from any provider. Since providers are COM objects, consumers can access them in any language (C++, Basic, Java, and so on).

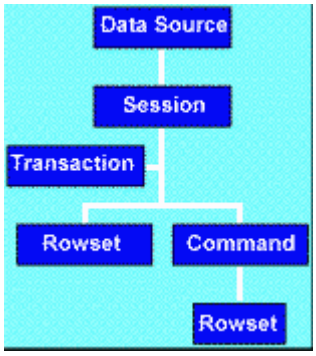
ADO is a lightweight object set for accessing data from any OLE DB provider, using a consistent API within any programming language and or environment.

The TrendWorX32 SQL Server application, which is ICONICS' next-generation enterprise data-logging and historical data server, uses relational databases as storage media to log process historical data. The historical data can be accessed through a variety of methods, such as:

- Using the OPC Historical Data Access interface which is supported in TrendWorX32 SQL Server
- Using the TrendWorX32 SQL Tool ActiveX Control to retrieve process data in a VBA-scripting environment
- Using the TrendWorX32 Reporting, which creates a variety of historical data reports in databases, .csv and .txt files, and Excel
- Using ADO within any programming environment to parse TrendWorX32 SQL Server historical tables

The TrendWorX32 OLE DB Provider, which encapsulates all the work needed to retrieve historical data spanning multiple historical data tables, interpolate and or extrapolate for missing values, and perform custom averaging of data over desired time periods.

OLE DB providers are a set of COM objects that transfer data from a durable source to a consumer. The OLE DB provider places those data in a tabular format in response to calls from a consumer. Providers can be simple or complex. A provider may return a table, allows the client to determine the format of that table, or perform operations on those data. Each provider implements a standard set of COM objects to handle requests from the client. A provider may implement optional COM objects to provide additional functionality. The figure below shows the different COM objects a provider may implement.



COM Components in an OLE DB Provider

COM Components and Their Uses

Component	Interfaces	Comments
Data Source	[mandatory] **IDBCreateSession** [mandatory] **IDBInitialize** [mandatory] **IDBProperties** [mandatory] **IPersist** [optional] **IDBDataSourceAdmin** [optional] **IDBInfo** [optional] **IPersistFile** [optional] **ISupportErrorInfo**	Connection from the consumer to the provider. The object is used to specify properties on the connection, such as user ID, password, and data source name. The object can also be used to administer a data source (create, update, delete, tables, and so on).
Session	[mandatory] **IGetDataSource** [mandatory] **IOpenRowset** [mandatory] **ISessionProperties** [optional] **IDBCreateCommand** [optional] **IDBSchemaRowset** [optional] **IIndexDefinition**	The session object represents a single conversation between a consumer and provider. It is somewhat similar to the ODBC HSTMT in that many sessions can be active simultaneously. The session object is the primary link to get to OLE DB functionality. In order to get to a command, transaction, or

	[optional] **ISupportErrorInfo** [optional] **ITableDefinition** [optional] **ITransaction** [optional] **ITransactionJoin** [optional] **ITransactionLocal** [optional] **ITransactionObject**	rowset object, you go through the session object.
Command	[mandatory] **IAccessor** [mandatory] **IColumnsInfo** [mandatory] **ICommand** [mandatory] **ICommandProperties** [mandatory] **ICommandText** [mandatory] **IConvertType** [optional] **IColumnsRowset** [optional] **ICommandPrepare** [optional] **ICommandWithParameters** [optional] **ISupportErrorInfo**	The command object handles operations on data such as queries. It can handle statements with or without parameters. The command object is also responsible for handling bindings for parameters and output columns. A binding is a structure that contains information about how a column, in a rowset, should be retrieved. It contains information such as ordinal, data type, length, status, and so on.
Rowset	[mandatory] **IAccessor** [mandatory] **IColumnsInfo** [mandatory] **IConvertType** [mandatory] **IRowset** [mandatory] **IRowsetInfo** [mandatory] **IRowsetIdentity** [optional] **IColumnsRowset** [optional] **IConnectionPointContainer** [optional] **IRowsetChange** [optional] **IRowsetLocate** [optional] **IRowsetResynch** [optional] **IRowsetScroll** [optional] **IRowsetUpdate** [optional] **ISupportErrorInfo**	The rowset object represents the data from the data source. The object is responsible for the bindings of that data and any basic operations (update, fetch, movement, and so on) on the data. You will always have a rowset object to contain and manipulate data.
Transaction	[mandatory] **IConnectionPointContainer**; [mandatory] **ITransaction** [optional] **ISupportErrorInfo**	The transaction object defines an atomic unit of work on a data source and determines how those units of work relate to each other. This object is not directly supported by the OLE DB provider templates. That is, you create your own object.

Each COM component represents a series of COM interfaces. Some COM interfaces are mandatory, while others are optional. By implementing the mandatory interfaces, a provider guarantees a minimum level of functionality that any client should be able to use. By implementing the optional interfaces, a provider can have more functionality and a richer feel to the client. The client should always call "QueryInterface" to determine whether a provider supports a given interface.

The TrendWorX32 OLE DB Provider is a simple read-only provider, which implements all of the mandatory interfaces listed in the table above, with the exception of the transaction-based component. TrendWorX32 OLE DB Provider supports, among others, the following:

- Similar historical replay functionality to the OPC Historical Data Access specification
- Raw or time-processed data-retrieval over a user-defined time interval
- Multiple historical tag data retrieval
- Dedicated "trend SQL" query parsing and execution capability
- Historical database browsing/prompting capability
- Schema information support for the "Recordset" structure it returns
- Support for a "MaxRecords" property to limit the total number of returned samples
- Multiple historical data filter support

The TrendWorX32 OLE DB Provider internally uses a high-speed data-retrieval mechanism based on the native OLE DB providers for the historical databases configured in the TrendWorX32 SQL data-logging server.

Using the TrendWorX32 OLE DB Provider

In this section, we will discuss how the TrendWorX32 OLE DB Provider can be used to retrieve data from the TrendWorX32 SQL Server historical database capability.

Connecting to TrendWorX32 OLE DB Provider

To connect to the TrendWorX32 OLE DB Provider, you must set the corresponding properties or ask the provider to prompt and provide database navigation and browsing tools.

In order to have the provider prompt, a typical VBA type of code would be:

```
Dim twx As New ADODB.Connection

' Set Provider

twx.Provider = " ICONICS.TWXOLEDB "

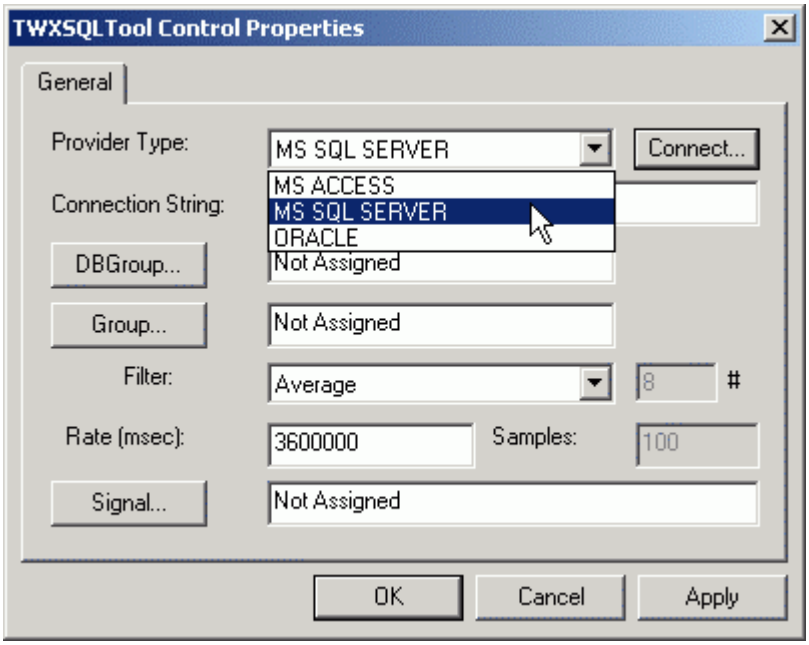
' Set up for Prompting

twx.Properties("Prompt") = 1

'Open Connection

twx.Open
```

Use the **TWXSQL Tool Control Properties** dialog box, shown below, to connect to a database. This dialog box has only one tab, the **General** tab shown below, which prompts you to complete the configuration.



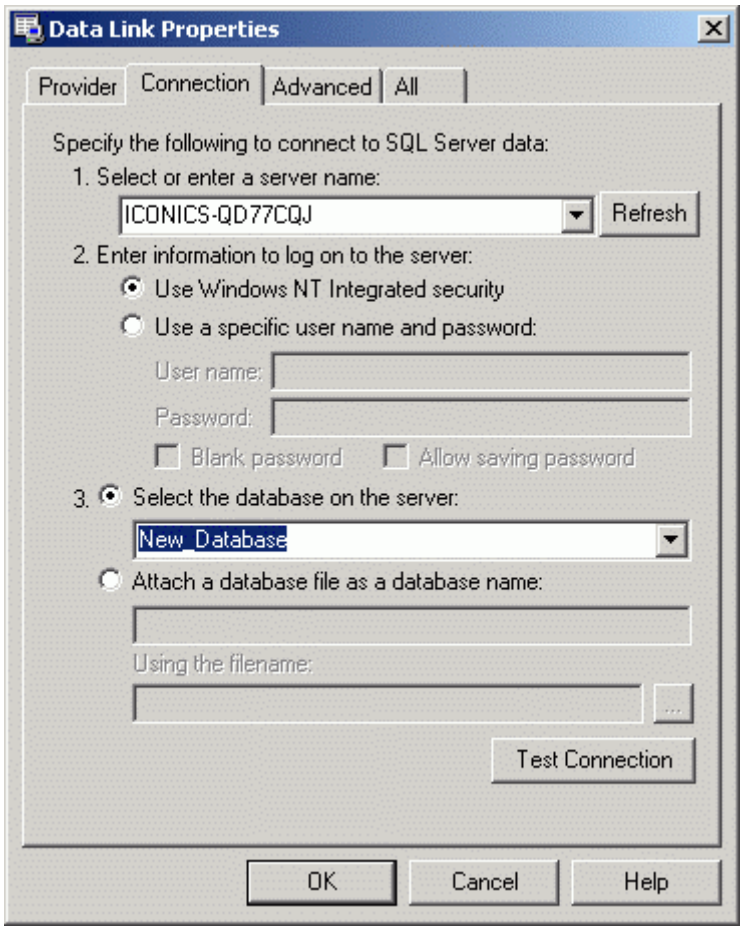
TWXSQL Tool Control Properties Dialog Box

To configure the SQL Tool control, from the **Provider Type** list box in the **TWXSQL Tool Control Properties** dialog box, select the OLE Database (DB) Provider type relating to the TrendWorX32 historical database. Currently, the following OLE DB Providers are supported:

- Microsoft Access
- Microsoft SQL Server
- Oracle
- My SQL

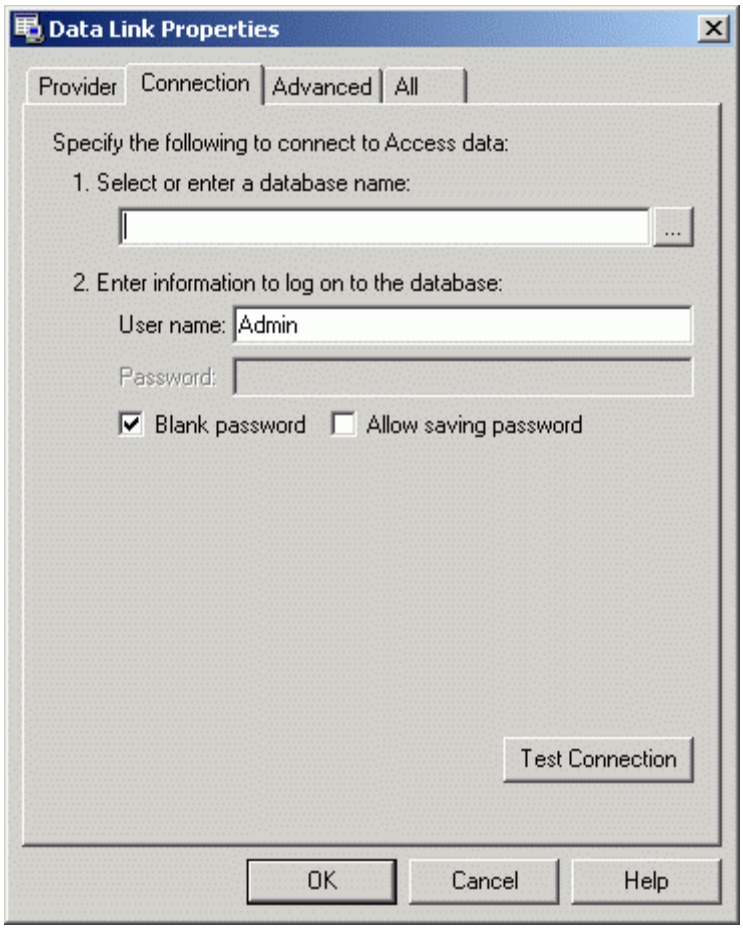
When you select a database type and then click **Connect,** the corresponding OLE DB Provider configuration dialog box opens, as shown below.

For Microsoft SQL Server, select a server name, an authentication type (Windows NT Integrated Security or user name and password), and the desired historical database, as shown in the figure below.



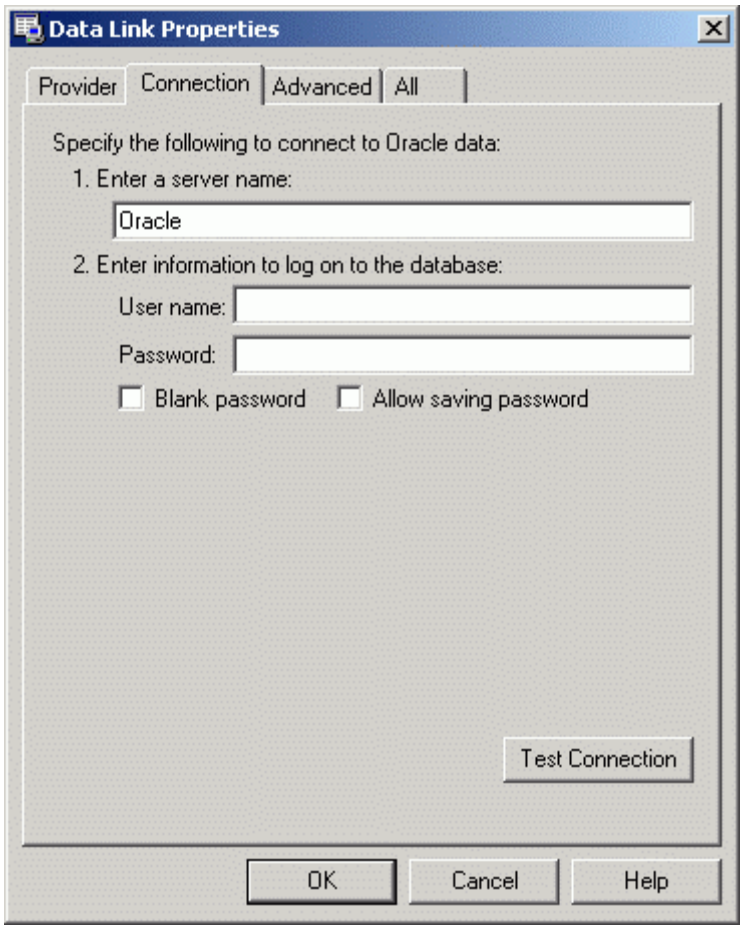
OLE DB Provider Configuration for SQL Server Databases

For Microsoft Access connections, simply navigate to the desired .mdb file by clicking on the browser button, as shown in the figure below.



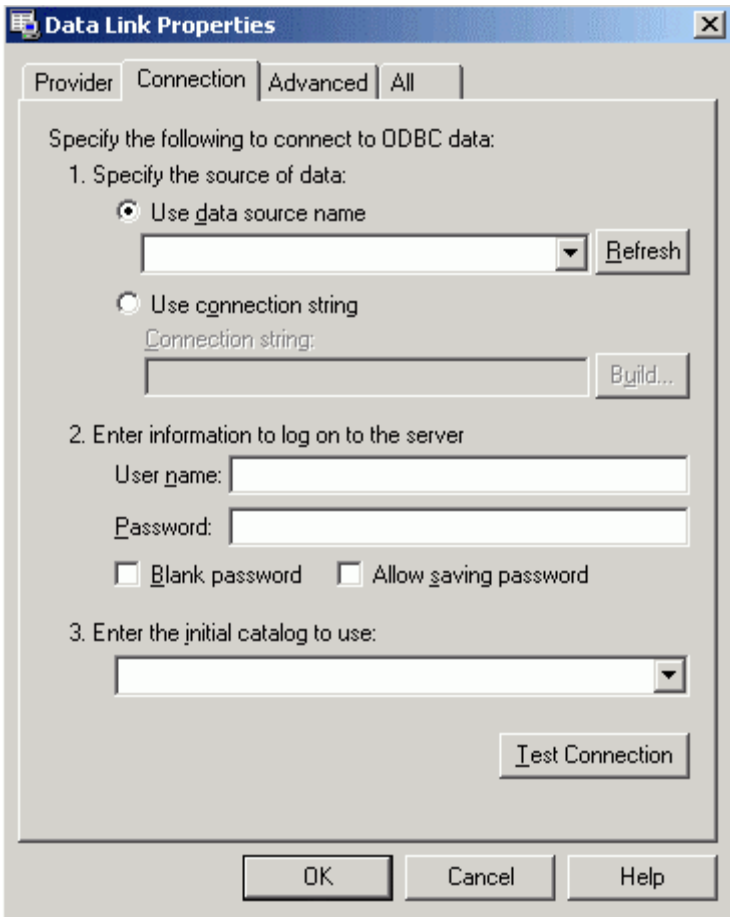
OLE DB Provider Configuration for Microsoft Access Databases

To connect to an Oracle database, fill in the appropriate information, as shown in the figure below.



OLE DB Provider Configuration for Oracle Databases

To connect to a MySQL database, specify the desired **Data Source Name,** the **Database Name, User, Password,** etc., as shown in the figure below.



OLE DB Provider Configuration for MySQL Databases

To increase speed and efficiency, the TrendWorX32 OLE DB Provider establishes a connection on a per-data-logging group basis and retrieves data for the historical tags in that group. You can always close an existing connection, set different properties and reopen the connection to retrieve data for a different group of historical tags.

Alternatively, you can set the required properties for opening a connection of the TrendWorX32 OLE DB Provider individually. These properties are:

Data source. For a Microsoft Access historical database, the data source is the complete path (including the file name) of the Microsoft Access .mdb file. For a Microsoft SQL Server historical database, the data source is the server name used for local SQL server installation. If the Microsoft SQL Server is installed locally, the server name will be "(local)."

Password. The required password. If the underlying historical database is a Microsoft SQL Server database, and no password is defined, the ICONICS TrendWorX32 OLE DB Provider will attempt to use NT integrated security.

User ID. The user ID name (if required).

Location. If the underlying historical database is a Microsoft SQL Server database, this property will be used for storing the initial catalog (i.e. the default SQL server database) for the connection to the historical database.

Extended properties. This is a dedicated ICONICS TrendWorX32 OLE DB Provider string that is formulated as follows:

TWXPROVIDER=provider;TWXDBASE=database group;TWXGROUP=group;

"Provider" is the internal OLE DB Provider name (i.e. SQL OLE DB for Microsoft SQL Server, or Microsoft.Jet.OLEDB.4.0 for Microsoft Access historical database)

"Database group" is the historical database group.

"Group" is the desired group of historical tags under the database group for which the provider will retrieve data.

Using the properties listed above, you can also connect to the ICONICS TrendWorX32 OLE DB Provider through ADO in the following ways:

```
Dim twx As New ADODB.Connection

' Set Provider

        twx.Provider = " ICONICS.TWXOLEDB "

' Set up Properties

twx.Properties("Prompt") = 0

twx.Properties("Data Source") = “(local)”

twx.Properties("Password")=””

twx.Properties("User ID")=””

twx.Properties("Extended Properties")=
”TWXPROVIDER=sqloledb;TWXDBASE=MyFirstDBase;TWXGROUP
=LoopX;”

twx.Properties("Location")=”TWXSQL_TEST_1”


'Open Connection

        twx.Open
```

This code assumes the following:

- The historical database is a Microsoft SQL Server database.
- A trusted connection is used.
- The SQL Server database where the historical data are logged is called "TWXSQL_TEST_1."
- The Microsoft SQL Server is located on the local workstation.

In addition to the above properties, the ICONICS TrendWorX32 OLE DB Provider supports language aliasing by making its resources available in a resource .dll. When translating the resources, do *not* translate the strings related to the default properties of the provider.

Retrieving Historical Data

TrendWorX32 Reporting has been updated to be compliant with the latest OPC HDA specification 1.2. Although TrendWorX32 Reporting is not a direct OPC HDA client or server, it creates reports with data outputs as specified by the OPC HDA specification.

Version 7.x introduced an enhanced data-retrieval system, which utilizes an updated approach to creating historical reports:

- Data filter selection other than Raw will result in data time stamped at the beginning of each subinterval.
- When retrieving data using data filters other than Raw, subintervals for which there are no data because of no data-logging activity will be marked as “empty” slots at the corresponding time with a zero value. You can check the returned qualities for further processing.
- The historical qualities supported are as follows:
  - OPCHDA_EXTRADATA 0x0001
    - More data may exist
  - OPCHDA_INTERPOLATED 0x0002
    - Interpolated data value
  - OPCHDA_RAW 0x0004
    - Raw data value
  - OPCHDA_CALCULATED 0x0008
    - Calculated data value
  - OPCHDA_NOBOUND 0x0010
    - No data found to provide upper or lower bound
  - OPCHDA_NODATA 0x0020
    - No data collected. Archiving not active
  - OPCHDA_DATALOST 0x0040
    - Calculation started/stopped/lost
  - OPCHDA_CONVERSION 0x0080
    - Scaling / conversion error
  - OPCHDA_PARTIAL 0x0100
    - Aggregate value is for an incomplete interval

Critical Note: Because of the new compliance changes in historical replay and reporting, a version 7.x and subsequent versions historical replay or reporting may not produce identical results to existing 6.1x version reports and or historical replay screen captures for the same signals and time periods.

The TrendWorX32 OLE DB Provider supports a minimal set of “trend SQL” keywords that can be used to formulate “trend” queries in order to retrieve historical data. A typical “trend” SQL statement may look like the following:

Select=Device1.Group.Tag1,Device1.Group.Tag10;Filter=MaxTime; SubInterval=10000; Start=1999/6/16, 18:00:00; End=1999/6/16, 18:10:00;

Select=tag1,tag2,…,tagn

This specifies the desired historical tags to participate in the query. If the `*' is used instead (e.g. Select=*,), without other keywords, the ICONICS TrendWorX32 OLE DB Provider will return information relating to the properties of the historical tags, such as device point name, logging name, and units. Note that in this query the logging names of the historical points should be used.

Filter=filter type;

This specifies the desired historical data-processing filter to be used when retrieving data historical data. The supported filter types are:

- Raw (all values are retrieved)
- Average
- Min
- Max
- StDev
- MinTime (minimum values with actual time stamp)
- MaxTime (minimum values with actual time stamp)
- Last (decimated value)
- All Statistics (maximum, minimum, standard deviation, and average over entire time span)
- Total (summary of values)
- Running Min
- Running Max
- Running Average (exponentially weighted moving average)
- Moving Average
- Moving Min
- Moving Max

If no "Filter" value is specified, the TrendWorX32 OLE DB Provider will retrieve all raw data-logged values.

SubInterval=interval;

This specifies the interval (in msec) between processed samples. The TrendWorX32 OLE DB Provider will process all samples, which have time stamps within a time interval to compute the corresponding sample with the desired filter. It is ignored if the data-retrieval filter is set to "Raw" or "AllStat." If it is set to 0, all historical samples within the search time interval will be included in creating a “processed” sample, which will be returned to the user.

Note that when the data filter is set to "Raw," the ICONICS TrendWorX32 OLE DB Provider will return all historical samples, which fall within the search time interval using the original time stamp that was data-logged. On the other hand, if it is not set to "Raw," the ICONICS TrendWorX32 OLE DB Provider will divide the search time interval into subintervals of "SubInterval" duration and “process” all samples that fall within each subinterval to calculate a “processed” sample to return.

Start=yyyy/mm/dd, hh:mm:ss;

This specifies the starting time for retrieving historical data, and it has to match the indicated format. If it is not specified, the time search will include all historical values starting at the beginning of the historical data

End=yyyy/mm/dd, hh:mm:ss;

This specifies the ending time for retrieving historical data, and it has to match the indicated format. If it is not specified, the time search will include all historical values up to the most recent ones

Important Notes

If any or both of the start and end dates are not specified, the TrendWorX32 OLE DB Provider will perform an exhaustive database search to retrieve all samples in the database, or those samples starting from the start date or ending prior to the end date. Because this can be an extremely time-consuming operation, it is suggested that you use the "MaxRows" property of the provider to establish a limit to the total number of retrieved samples.

Note: The Unicode version has enhanced language aliasing support, including automatic value scaling. You need to configure the ICONICS Language Server, as well as configure language aliases in the TrendWorX32 Configurator. Once all language aliases are configured, you can use the LCID property of the TrendWorX32 OLE DB Provider connection to set the desired language settings.

Returned Recordset Structure

The TrendWorX32 OLE DB Provider returns the following recordset structure (column names):

TagLoggingName	A variable length text column displaying the logging name of the tag.
TagPointName	A variable length text column displaying the actual point name of the tag.
TimeDate	A "date" type showing the time stamp.
Msecs	A "long" type indicating the msec accuracy of the time stamp.
Value	A "double" type indicating the value of the sample.
OPCDataQuality	A "long" type indicating the OPC data access quality of the sample.
OPCHDAQuality	A "long" type indicating the OPC data access quality of the sample.
QualityText	A variable-length "text" type indicating the quality string related to the combined data and historical quality values.
DataFilter	A variable-length "text" type indicating the data filter type, such as "Raw," used in retrieving the data.
EngUnits	A variable-length "text" type indicating the "EngUnits" of the tag.

Important Note

In the case that only the tags under the desired logging group are returned (that is, when Select=*) the "DataFilter" column will contain the data-logging filter for the tag. See the TrendWorX32 SQL Server documentation for more information.

A code sample for opening a recordset to return all tags data-logged in a particular group and to display their logging name might be as follows:

```
Dim rstwx As New ADODB.Recordset
        Dim strLoggingName As String

        ' Set Recordset properties
        rstwx.CursorType = adOpenForwardOnly
        rstwx.LockType = adLockReadOnly
        ' Set cache size
                rstwx.CacheSize = 100
        ' Open Recordset
                rstwx.Open "Select=*;", twx, , , adCmdText
```

```
‘ Process Records
Do While Not rstwx.EOF
    strLoggingName = rstwx!TagLoggingName
rstwx.MoveNext
Loop

' Close Recordset
If rstwx.State = adStateOpen Then
  rstwx.Close
End If

End Sub
```

In this case, "twx" is the ICONICS TrendWorX32 OLE DB Provider ADO connection object we just opened.

Example Application Using TrendWorX32 OLE DB Provider

In order to provide a sample application using the TrendWorX32 OLE DB Provider, has created a sample VBA application to demonstrate its use. It is located under the GENESIS32 "Examples" directory in the "TrendWorX32 Examples\TrendWorX Data Logging and Reporting" folder, and it is called "TrendOleDB.t32." This file consists of a main form, which is shown below. Click **Connect** to connect to a TrendWorX32 SQL Server historical database. Click **Signal Info** to retrieve the data-logged signal for a particular group. You can also build a trend query and execute it to retrieve historical data.

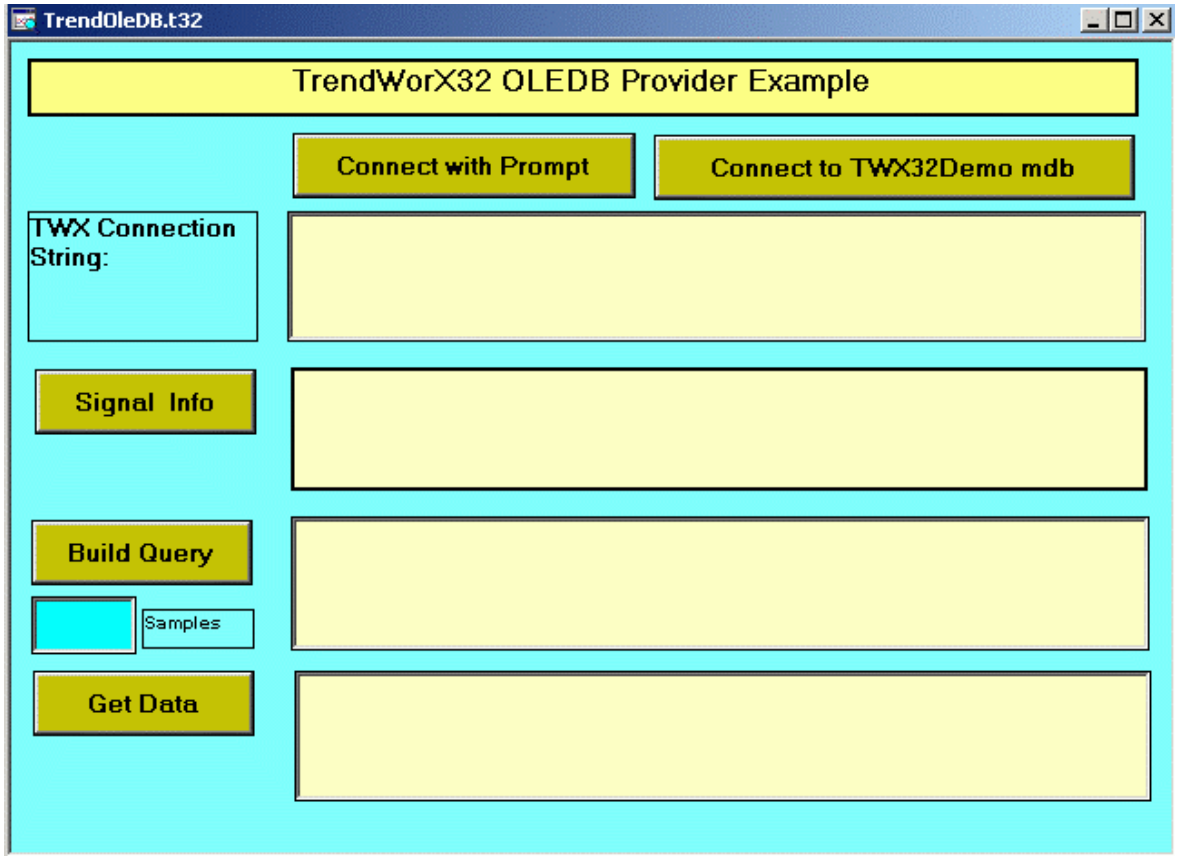
To prepare for the example, you must:

- Set the GENESIS32 working directory to be "TrendWorX32 Examples\TrendWorX Data Logging and Reporting," where the examples are installed. (For example you can use the **Set Working Directory** command in the **Tools** menu of the TrendWorX32 Container.
- Make sure that none of the files in this folder has a read-only attribute.

Once you have completed these steps, do the following:

1. Start the TrendWorX32 Container.
2. Load the TrendOleDB.t32 display into TrendWorX32.
3. Enter animation mode.
4. Click the various buttons to retrieve TrendWorX32 historical data using the TrendWorX32 OLE DB Provider.
5. When prompted, you can select Microsoft Access as the historical database type and enter "Twx32Demo.mdb."

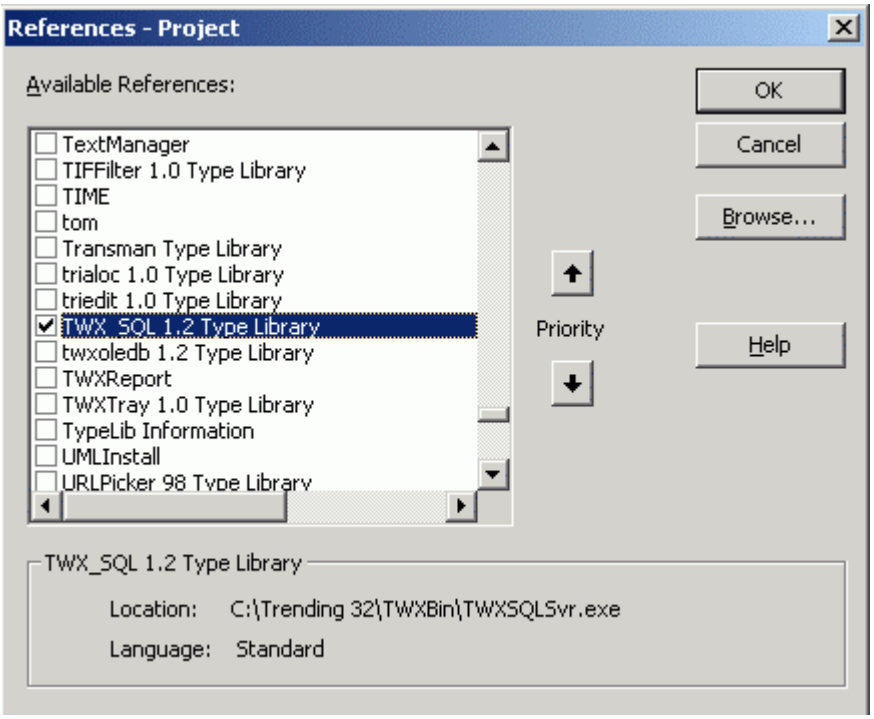
6. You can now start data logging, stop data logging, and retrieve historical data.



Sample Visual Basic Application for the TrendWorX32 OLE DB Provider

TrendWorX32 SQL Data Logger OLE Automation Reference

The TrendWorX32 SQL Data Logger supports a dual COM interface for data logging manipulation through OLE Automation or DCOM. To use this new interface within a Visual Basic application (VBA), you must ensure that the VBA project has **References** to the ICONICS TrendWorX32 SQL library, as shown below.



Adding References to the TWX_SQL Type Library

Then you can use the methods and properties exposed. A simple call to create an instance of the server object within Visual Basic is as follows:

```
Dim mylog As TWXLogger

Set mylog = CreateObject("Iconics.TWXLogger.1")
```

The Data Logger exposes the following OLE automation methods and properties:

```
[propget, id(1), helpstring("property LoggerStatus")]

HRESULT LoggerStatus([out, retval] short *pVal);


[propget, id(2), helpstring("property CurrentTime")]

HRESULT CurrentTime([out, retval] DATE *pVal);
```

```
[propget, id(3), helpstring("property StatusString")]
HRESULT StatusString([out, retval] BSTR *pVal);

[propget, id(4), helpstring("property StartTime")]
HRESULT StartTime([out, retval] DATE *pVal);

[id(5), helpstring("method GetDatabaseGroupStatus")]
HRESULT GetDatabaseGroupStatus([in] BSTR bstrDBaseGroupName,
                [out] short * pStatus,
                [out] DATE * pCurrentTime,
                [out] BSTR * bstrStatusString );

[id(6), helpstring("method GetGroupStatus")]
HRESULT GetGroupStatus([in] BSTR bstrDBaseGroupName,
                [in] BSTR bstrGroupName,
                [out] short * pStatus,
                [out] DATE * pCurrentTime,
                [out] BSTR * bstrStatusString);

[id(7), helpstring("method GetGroupStatistics")]
HRESULT GetGroupStatistics([in] BSTR bstrDBaseGroupName,
                [in] BSTR bstrGroupName,
                [out] long * pConnectionTime,
                [out] long * pTableOpenTime,
                [out] long * pDataPreparationTime,
                [out] long * pDataWriteTime,
                [out] long * pTotalLogTime,
                [out] long * pTotalLogMinTime,
                [out] long * pTotalLogMaxTime,
                [out] long * pTableOpenMinTime,
```

```
                [out] long * pTableOpenMaxTime,
                [out] long * pTotalAvgWriteTime,
                [out] DATE * pLastWriteTime,
                [out] BSTR * bstrTableName);

    [id(8), helpstring("method LoadConfiguration")]
    HRESULT LoadConfiguration([in] long lConfigID,
                [in] BSTR bstrConfigurationName);

    [id(9), helpstring("method SetConfigurationDatabase")]
    HRESULT SetConfigurationDatabase([in] BSTR bstrConnectionString);

    [id(10), helpstring("method StartLogging")]
    HRESULT StartLogging();

    [id(11), helpstring("method StopLogging")]
    HRESULT StopLogging();

    [id(12), helpstring("method ActivateDatabaseGroup")]
    HRESULT ActivateDatabaseGroup([in] BSTR bstrDBaseGroupName);

    [id(13), helpstring("method DeActivateDatabaseGroup")]
    HRESULT DeActivateDatabaseGroup([in] BSTR bstrDBaseGroupName);

    [id(14), helpstring("method ActivateGroup")]
    HRESULT ActivateGroup([in] BSTR bstrDBaseGroupName,
                [in] BSTR bstrGroupName);

    [id(15), helpstring("method DeActivateGroup")]
    HRESULT DeActivateGroup([in] BSTR bstrDBaseGroupName,
```

```
                [in] BSTR bstrGroupName);

  [id(16), helpstring("method Switch Database")]
  HRESULT SwitchDatabase([in] BSTR bstrDBaseGroupName,
                [in] BSTR bstrODBCConnString,
                [in] BSTR bstrOLEDBConnString,
                [in] short nDBType,
                [in] BOOL bReDirect);

  [id(17), helpstring("method ReloadConfiguration")]
  HRESULT ReloadConfiguration([in] long lConfigID,
                [in] BSTR bstrConfigurationName);

 [id(18), helpstring("method AdjustLoggingParameters")]
  HRESULT AdjustLoggingParameters([in] DWORD dwMaxRetrials,
                [in] DWORD dwMaxPackets);

  [id(19), helpstring("method GetCurrentConfiguration")]
  HRESULT GetCurrentConfiguration([out] long * lConfigID,
                [out] BSTR * bstrConfigurationName,
                [out] BSTR * bstrConnectionString);

  [id(20), helpstring("method ResetConfiguration")]
  HRESULT ResetConfiguration();

  [id(21), helpstring("method TestConnection
  HRESULT TestConnection([in] BSTR bstrConnectionString
                );

  [id(22), helpstring("method AdjustLoggingParameters2")]
```

```
HRESULT AdjustLoggingParameters2(
        [in] DWORD dwMaxRetrials,
        [in] DWORD dwMaxPackets,
        [in] DWORD dwMaxSamples,
        [in] DWORD dwRetrialDelay,
        [in] BOOL  bUseGlobalConnection
        );

[id(23), helpstring("method QueryLoggingParameters")]
HRESULT QueryLoggingParameters(
        [out] DWORD * dwMaxRetrials,
        [out] DWORD * dwMaxPackets,
        [out] DWORD * dwMaxSamples,
        [out] DWORD * dwRetrialDelay,
        [out] BOOL  * bUseGlobalConnection
        );
```

In addition, the following enumerations are used:

Data Logger status:

- **TWXSQL_UP = 1**
- **TWXSQL_DOWN**
- **TWXSQL_INDETERMINATE**

Database group status:

- **TWXSQL_DBASEGROUP_ACTIVATED = 1**
- **TWXSQL_DBASEGROUP_DEACTIVATED**
- **TWXSQL_DBASEGROUP_INDETERMINATE**

Logging group status:

- **TWXSQL_GROUP_ACTIVATED = 1**

- **TWXSQL_GROUP_DEACTIVATED**
- **TWXSQL_GROUP_STANDBY**
- **TWXSQL_GROUP_EXITING**
- **TWXSQL_GROUP_INDETERMINATE**

Database connection type:

- **TWXSQL_DBASE_NONE = 1,**
- **TWXSQL_DBASE_ACCESS**
- **TWXSQL_DBASE_SQLSVR**
- **TWXSQL_DBASE_ORACLE**

In the method calls listed above:

- **bstrDBaseGroupName:** Refers to the database group name, as it would appear in the Configurator database.
- **bstrGroupName:** Refers to the logging group name, as it would appear in the Configurator database.
- **lConfigID:** Refers to the configuration ID, as it would appear in the Configurator database.
- **bstrConfigurationName:** Refers to the configuration name, as it would appear in the Configurator database, corresponding to the "lConfigID."
- **bstrODBCConnString:** Refers to the ODBC connection string passed to the OLE DB provider for ODBC, as it would appear in the Configurator database.
- **bstrOLEDBConnString:** Refers to the OLE DB connection string passed to the OLE DB provider for Microsoft Access, Microsoft SQL Server (MSDE), or Oracle, as it would appear in the Configurator database.
- **bstrConnectionString:** Refers to the OLE DB connection string passed to the OLE DB provider for Microsoft Access used in establishing a configuration database connection for the TrendWorX32 SQL Data Logger.

Note: The "SwitchDatabase" method call can be used while during data logging to perform a hot database switch operation with the option to redirect pending writes to the new database connection. If the Data Logger cannot establish a database schema information on the new database, or the new database type requires more Client Units, which are not available due to the current level of ICONICS Licensing, this operation will fail.

OPC HDA COM Interface

The TrendWorX32 SQL Data Logger Version 7.x was updated to be compliant with the latest OPC HDA specification. In general, the OPC HDA specification is an extended OPC specification, which includes an increased number of interfaces and methods, not all of them being mandatory.

Critical Note: Because of the compliance changes in historical replay and reporting, version 7.x or later historical replay or reporting may not produce identical results to existing 6.1x version reports and or historical replay screen captures for the same signals and time periods.

In addition to the Data Logger interface, the TrendWorX32 SQL Data Logger also implements the OPC Historical Data Access (HDA) interface. The following interfaces and methods are supported, including connection point support. Version 7.x included an enhanced data-retrieval system, which utilizes a new approach to creating historical reports. The following is a summary of the changes for version 7.x:

- Data filter selection other than Raw will result in data time stamped at the beginning of each subinterval.
- When retrieving data using data filters other than Raw, subintervals for which there are no data because of no data-logging activity will be marked as “empty” slots at the corresponding time with an empty (VT_EMPTY) value. You can check the returned qualities for further processing.
- The historical qualities supported are as follows:
    - OPCHDA_EXTRADATA 0x0001
        - More data may exist
    - OPCHDA_INTERPOLATED 0x0002
        - Interpolated data value
    - OPCHDA_RAW 0x0004
        - Raw data value
    - OPCHDA_CALCULATED 0x0008
        - Calculated data value
    - OPCHDA_NOBOUND 0x0010
        - No data found to provide upper or lower bound
    - OPCHDA_NODATA 0x0020
        - No data collected. Archiving not active
    - OPCHDA_DATALOST 0x0040
        - Calculation started/stopped/lost
    - OPCHDA_CONVERSION 0x0080
        - Scaling / conversion error
    - OPCHDA_PARTIAL 0x0100
        - Aggregate value is for an incomplete interval

Interfaces

The following OPC HDA COM interfaces are supported:

- **interface IOPCHDA_Server**
- **interface IOPCHDA_Browser**
- **interface IOPCHDA_SyncRead**
- **interface IOPCHDA_AsyncRead**
- **interface IOPCHDA_SyncAnnotations**

Methods

The TrendWorX32 SQL Server supports the following OPC HDA Interfaces and methods:

- **interface IOPCHDA_Server**
  - ✓ HRESULT GetItemAttributes
  - ✓ HRESULT GetAggregates
  - ✓ HRESULT GetHistorianStatus
  - ✓ HRESULT GetItemHandles
  - ✓ HRESULT ReleaseItemHandles
  - ✓ HRESULT ValidateItemIDs
  - ✓ HRESULT CreateBrowse

- **interface IOPCHDA_Browser**
  - ✓ HRESULT GetEnum
  - ✓ HRESULT ChangeBrowsePosition
  - ✓ HRESULT GetItemID
  - ✓ HRESULT GetBranchPosition

- **interface IOPCHDA_SyncRead**
  - ✓ HRESULT ReadRaw
  - ✓ HRESULT ReadProcessed
  - ✓ HRESULT ReadAtTime
  - ✓ HRESULT ReadModified

- ✓ HRESULT ReadAttribute

- **interface IOPCHDA_AsyncRead**
  - ✓ HRESULT ReadRaw
  - ✓ HRESULT ReadProcessed
  - ✓ HRESULT ReadAtTime
  - ✓ HRESULT ReadModified
  - ✓ HRESULT ReadAttribute
  - ✓ HRESULT Cancel

- **interface IOPCHDA_SyncAnnotations**
  - ✓ HRESULT QueryCapabilities
  - ✓ HRESULT Read
  - ✓ HRESULT Insert

TrendWorX32 Viewer ActiveX OLE Automation Reference

This section describes the OLE Automation interfaces for the TrendWorX32 Viewer.

Interfaces

Each ActiveX exposes a set of objects, which are used to configure and automate the control. Each object categorizes an area of functionality. The following objects are exposed by the TrendWorX32 Viewer ActiveX:

- **IOTitle**
- **IORanges**
- **IOTimeSettings**
- **IODetails**
- **IOPen**
- **IOGrids**

Properties and Methods for the TrendWorX32 Viewer ActiveX

The following properties and methods are categorized by the object in which they exist.

Viewer: Properties

boolean AutoStartRuntime	Toggles the runtime mode when the container application switches from design mode to active mode. Default: TRUE
boolean ShowTitle	Determines whether the main title is displayed. Default: FALSE
boolean ShowRanges	Determines whether the ranges field is displayed. Default: TRUE
boolean ShowDetails	Determines whether the details field is displayed. Default: TRUE
BSTR PlotType	Determines plot type. Values are: time plot, XY plot, logarithmic plot, strip chart, bar plot, and circular chart. Default: time plot
BSTR BorderEdge	Determines trend border area settings. Values are: sunken, raised, etched, and bumped. Default: Sunken
BSTR BorderOptions	Determines trend border area options. Values are: rectangle, left, bottom, right, top, top left, top right, bottom left, and bottom right. Default: Rectangle
OLE_COLOR DispColor	Determines trend curve area color.
OLE_COLOR BackColor	Determines background color.

LPFONTDISP TimeFont	Dispatches interface to the time field font.
LPFONTDISP DetailsFont	Dispatches interface to the detail field font.
LPFONTDISP MainFont	Dispatches interface to the main (title) font.
boolean ShowTimeInfo	Determines if the time field is displayed. Default: TRUE
boolean MainToolbar	Enables/disables **Trend Viewer** toolbar. Default: TRUE
boolean FreezeMode	Enables/disables freeze mode.
short NumberOfPens	Returns number of currently configured pens.
LPFONTDISP RangesFont	Dispatch interface to the range field font.
BSTR FileName	Gets/sets file name for saving configuration data. "Set" is not supported while in runtime mode.
boolean URLPathUsed	Enables/disables use of URL path for asynchronous downloads. Default: FALSE
BSTR URLPathName	Gets/sets URL path name.
BSTR StatisticsFile	Gets/sets the .csv/.txt file used for statistical information storage. "Set" is not supported while in runtime mode.
boolean StatFileAppend	Enables/disables append mode for statistical file. Default: FALSE
boolean GlobalRanges	Enables/disables the use of global

	ranges. Not supported in runtime (animation) mode. Default: FALSE
boolean ShiftGrids	Enables/disables the shifting of grids during runtime. Not supported in runtime (animation) mode. Default: TRUE
double StartUp	Gets/sets initial offset for plotting of curves. Default: 0
short CursorStep	Gets/sets the cursor step for stepping in freeze mode. Default: 10
LPDISPATCH Title	Dispatch interface to Title object.
LPDISPATCH Details	Dispatch interface to Details object.
LPDISPATCH Ranges	Dispatch interface to ranges object.
LPDISPATCH TimeSettings	Dispatch interface to Time Settings object.
LPDISPATCH XGrids	Dispatch interface to X Grids object.
LPDISPATCH YGrids	Dispatch interface to Y Grids object.
boolean RestoreRanges	Enables/disables restoring of ranges modified during runtime. Not supported in runtime (animation) mode Default: FALSE
long LanguageID	Gets/sets the Language ID for loading a multilingual resource .dll.
boolean FillBackColor	Enables/disables the color fill of trend display areas.

	This should be used prior to and after printing (within the container) to conserve color ink on the printer.
long ScrollBarRange	Sets the range of the scrollbar (in seconds). Default: 10,800 seconds (3 hours)
boolean FullChartReset	Resets Circular Chart after a full trend period elapses. "Set" is not supported while in runtime mode. Default: FALSE
long ReadyState	Signals controls state changes in loading configuration data. Used for asynchronous download of configuration data
short Appearance	Determines TrendWorX32 Viewer ActiveX appearance (3D edge or not). Default: TRUE
short BorderStyle	Determines TrendWorX32 Viewer ActiveX appearance (border or not). Default: TRUE
OLE_COLOR ForeColor	Determines title color and main display text color.
boolean UseLocalSettings	Sets the Local Settings support. If the TrendWorX32 Viewer is configured to save its configuration into a .v32 file, independently of the container, you will have the option in runtime mode to save current settings into the preconfigured file. In the case of a Web HMI application, this will also be the configuration file to be loaded whenever the HTML page is refreshed. Default: FALSE
BOOL PrintBkg	Toggles the background printing.
BOOL OldPlotStyle	Toggles the old plot style.
BOOL ShowBackgnd	Toggles the background image.

Viewer: Methods

boolean AddRealTimePen(BSTRPenName)	Adds a real-time pen, which can be an OPC data tag, expression tag, and internal simulation point. For OPC data tags, the "PenName" has to match exactly the string that the Tag Browser would produce for the same tag.
boolean SetXAxis (BSTR PenName)	Used in XY plots to select a pen for *x*-axis. The "PenName" here represents the description text for the desired pen, as it would appear in the **Details** window.
VARIANT GetPenNames()	Returns a variant string array of all pen description text as it appears in the **Details** window of the Viewer ActiveX.
boolean EnterAnimate()	Enters runtime (Animate) mode.
boolean ExitAnimate()	Exits runtime (Animate) mode.
boolean GetAnimateMode()	Returns TRUE if the Viewer is currently in animation mode. Otherwise it returns FALSE.
boolean DeletePen (BSTR PenName, BSTR PenType)	Deletes the pen that has "PenName" as its description text. Valid types are: real time, history, and custom.
void SetPenHiRangeString (BSTR PenName, BSTR RangeString, BSTR PenType)	Sets the point name for high range of the pen with description text matching "PenName." Not supported for historical pens
void SetPenLoRangeString (BSTR PenName, BSTR RangeString, BSTR PenType)	Sets the point name for low range of the pen with description text matching "PenName." Not supported for historical pens
LPDISPATCH GetPen (BSTR PenName, BSTR PenType)	Returns dispatch interface of the pen with description text matching "PenName."

	You must release the interface when done. Valid types are: real time, history, and custom.
boolean SetDetailsFromPen (BSTR PenName, BSTR PenType)	Populates details information from the pen with description text matching "PenName." Valid types are: real time, history, and custom.
boolean ComputeStat()	Computes and stores statistical info for the pens configured in the Viewer ActiveX. Supported only in runtime (animation) mode.
void SetHiRangeString (BSTR RangeString)	Sets point name for high range of range object. Not supported in history-only mode.
void SetLowRangeString (BSTR RangeString)	Sets point name for low range of range object. Not supported in history-only mode.
boolean ReplacePoint(BSTR PointName, BSTR OldString, BSTR NewString, boolean DoPenText)	Use to replace pen point names and pen descriptions if desired ("DoPenText" set to TRUE). "PointName" is the actual (existing) point name. "OldString" is a substring of the "PointName." "NewString" is its substitute string, and if "DoPenText" is set to TRUE, the corresponding pen description text will be replaced as well. Not supported in runtime (animation) mode.
void SetPointValue(BSTR PointName, double PointValue, DATE PointTDate)	Use to feed custom data for an assigned pen. "PointName" is the full point name by which the pen connection was made. Supported only in runtime (animation) mode.
void ChangeWinPeriod (long NewPeriod)	Changes the trend period into the desired "NewPeriod," given in seconds. Not supported for bar plots and XY plots.

boolean SetTrendRightTDate (DATE RightTDate)	In freeze mode, sets the trend's right time and date.
boolean AddRealTimeBgPen(BSTR PenName, BSTR TWXBgGroupName)	Adds a real-time pen from Persistent Trending, which can be an OPC data tag, expression tag, and internal simulation point. For OPC data tags, the "PenName" has to match exactly the string that the Tag Browser would produce for the same tag. "TWXBgGroupName" is the name of the configured Persistent Group.
void DeleteAllPens ()	Deletes all pens from the TrendWorX32 Viewer ActiveX.
boolean AddCustomPen (BSTR PenName)	Adds a custom pen for displaying custom data. You are responsible for supplying data to be displayed by the pen, using the "SetPenValues" method.
Boolean SetPenValues (BSTR PenName, VARIANT* PenValues, VARIANT* PenTDates, VARIANT* PenMSecs, VARIANT* PenQuality)	Fills a custom pen with user data. "PenName" is the description text of the pen, as it appears in the **Details** window. "PenValues" is an array of doubles. "PenTDates" is an array of date values corresponding to the "PenValues." "PenMSecs" is an array of milliseconds subdivisions corresponding to the "PenTDates" values. "PenQuality" is an array of OPC-defined qualities.
DATE GetRightTime()	Returns the Viewer ActiveX "right" time and date. Supported only in runtime (animation) mode.
boolean SetHistTrendMode(Boolean Mode)	Toggles history-only mode of the Viewer ActiveX. Supported only in configuration mode.
boolean GetHistTrendMode()	Retrieves the status of history replay mode of the Viewer ActiveX.

boolean AddHistoryPen(BSTR PenName)	Adds a history pen to the Viewer ActiveX. The "PenName" string should be the same as the one produced by the Tag Browser when browsing historical servers.
boolean UpdateHistoryPens()	Forces an update of all historical signals. Typically this would be called after setting the Viewer's right reference time. Supported only in freeze mode
boolean SetPenFilter(BSTR PenName, short PenFilter)	Sets the data filter type for a pen with description text matching "PenName." Used for historical pens. Valid "PenFilter" types are: OHDA_RAW = 0 OHDA_AVG = 1 OHDA_MIN = 2 OHDA_MAX = 3 OHDA_STDEV= 4 OHDA_LAST = 5 OHDA_MINTIME = 6 OHDA_MAXTIME = 7
void RefreshDetails()	Forces a refresh of the **Details** window. Since each pen is an individual object, any change in the pen's properties through OLE Automation may not be directly updated in the **Details** window. Using this ensures that all pen property changes will be reflected properly in the **Details** window. It is supported only in runtime (animate) mode
void ShowStat()	Shows the statistical window of the Viewer. Supported only in runtime (animation) mode.
boolean DoZoom(short nPercent, boolean bYaxis, boolean bXaxis)	Zoom in on the Viewer. If both "bYAxis" and "bXAxis" are set, then the zoom action will be applied to both axes. For an ordinary time plot, this means both range and time zooming. The "nPercent" parameter controls the zoom factor. If set to 0 (zero), it zooms out of the Viewer. Supported only in freeze mode.

boolean LoadSaveFile(BSTR FileName, short Load)	Loads or saves a Viewer ActiveX configuration file.
void SetFirstUpdateMode(boolean bMode)	If "bMode" is TRUE, in freeze mode the Viewer ActiveX will not update historical pens upon connection. You will have to call the "UpdateHistoryPens" method.
long GetPenCount()	Returns the number of configured pens.
LPDISPATCH GetPenByIndex(long PenIndex)	Returns the pen object, where "PenIndex" is the order of the pen as it appears in the **Details** window.
boolean ShowPenInRanges(long PenIndex)	Fills in the ranges display with the ranges of the pen, where "PenIndex" is the order of the pen as it appears in the **Details** window. Not supported if **Global Ranges** option is enabled.
long AddRealTimePen2(BSTR PenName)	Adds a real-time pen, which can be an OPC data tag, expression tag, and internal simulation point. For OPC data tags, the "PenName" has to match exactly the string that the Tag Browser would produce for the same tag. It returns the index of the pen added.
long AddHistoryPen2(BSTR PenName)	Adds a history pen to the Viewer ActiveX. The "PenName" string should be the same as the one produced by the Tag Browser when browsing historical servers. It returns the index of the pen added.
long AddRealTimeBgPen2(BSTR PenName, BSTR TWXBgGroupName)	Adds a real-time pen from Persistent Trending, which can be an OPC data tag, expression tag, and internal simulation point. For OPC data tags, the "PenName" has to match exactly the string that the Tag Browser would produce for the same tag. "TWXBgGroupName" is the name of the configured Persistent Group. It returns the index of the pen added.
long AddCustomPen2 (BSTR PenName)	Adds a custom pen for displaying custom data. It returns the index of the pen added.

boolean SetPenTimeShift(long PenIndex, long TimeShift)	Sets the time shift for the pen with index "PenIndex" by "TimeShift" seconds. Not supported for XY plots. Supported only in freeze mode.
long GetPenTimeShift(long PenIndex)	Returns the time shift for the pen with index "PenIndex" in seconds. Not supported for XY plots. Supported only in freeze mode.
boolean RefreshIdealPen(long PenIndex)	Refreshes the ideal pen, with index "PenIndex," with new historical data based on its ideal pen settings.
boolean DeletePen2(long PenIndex)	Deletes the pen with index "PenIndex."
boolean SetPenFilter2(long PenIndex, short PenFilter)	Sets the data filter type for a pen with index "PenIndex." Used for historical pens. Valid "PenFilter" types are: OHDA_RAW = 0 OHDA_AVG = 1 OHDA_MIN = 2 OHDA_MAX = 3 OHDA_STDEV= 4 OHDA_LAST = 5 OHDA_MINTIME = 6 OHDA_MAXTIME = 7
void ToggleMainToolbar()	Toggles **Trend Viewer** toolbar if enabled. Supported only in runtime (animation) mode.
void ToggleMainToolbar2(long top, long left)	Toggles **Trend Viewer** toolbar if enabled and positions it at the top, left coordinates. Supported only in runtime (animation) mode.
void AboutBox().	Displays the Viewer's About Box.
void Refresh()	Standard refresh method. Call after visual modifications.

boolean PrintTrend(boolean bShowPrinterDlg)	Prints the TrendWorX32 Viewer display. If "bShowPrinterDlg" is set to TRUE, the Windows **Print** dialog box will be evoked. Otherwise, the default Windows printer will be used.
boolean SetInkSaveMode(boolean bMode)	Sets the ink saving mode during printouts. If TRUE, during the printing process all of the Viewer background area colors will be ignored to save ink.
boolean GetInkSaveMode()	Returns the current status of the ink-saving mode.
boolean ResetPenTotalizer(long PenIndex)	Resets a pen's Totalizer function support. The "PenIndex" is the order of the pen.
void SetUTCReplayMode(boolean bMode)	Sets the UTC historical replay mode. Supported only for a history chart Viewer. If TRUE, all curves are drawn using UTC time stamping.
boolean GetUTCReplayMode()	Returns TRUE if the Viewer is replaying historical data in UTC mode.
boolean GetSampleInfo(long PenIndex, VARIANT* SampleValue, DATE* SampleTDate, VARIANT* SampleMSecs, VARIANT* SampleQual, VARIANT* SampleHDAQual, VARIANT* HistoricalSample)	Returns sample information for the current location of the cursor in freeze mode or zoom mode for the pen with index "PenIndex."
boolean GetSampleInfo2(long PenIndex, VARIANT* SampleValue, VARIANT* SampleTDate, VARIANT* SampleMSecs, VARIANT* SampleQual, VARIANT* SampleHDAQual, VARIANT* HistoricalSample);	Returns sample information for the current location of the cursor in freeze mode or zoom mode for the pen with index "PenIndex.". It returns the date as variant
void SetCircularPrintMode(boolean bAutoPrint)	Sets the automatic print mode of circular charts upon a full trend period reset. If "bAutoPrint" is set to TRUE, the Viewer will print trend display upon full chart reset.

boolean GetCircularPrintMode()	Returns the current automatic print mode of circular charts.
void MoveCursorForward()	In freeze mode, moves the cursor one step forward in time.
void MoveCursorBackwards()	In freeze mode, moves the cursor one step backward in time.
boolean MoveCursorToTime(DATE timeDate)	In freeze mode, moves the cursor to a particular position on the time axis.
long ChangeTrendDataRate(long trendDataRate)	Changes Viewer data-collection rate in animation (runtime) mode. "TrendDataRate" is the desired data-collection rate in milliseconds.
boolean GetPenValues(BSTR PenName, VARIANT* PenValues, VARIANT* PenTDates, VARIANT* PenMSecs, VARIANT* PenQuality)	Returns in variant arrays the current samples in memory buffers of the pen with point name (or description) "PenName."
void SetPenComments(long PenIndex, boolean Mode)	Sets the historical comment and batch information mode of a historical pen with index "PenIndex." If "Mode" is TRUE, it enables the display of historical comments.
boolean GetPenComments(long PenIndex, VARIANT* SourceTDates, VARIANT* SourceMsec, VARIANT* Notes, VARIANT* NoteTDates, VARIANT* NoteMSecs, VARIANT* Authors)	Retrieves the associated (if any) historical operator comments and batch information with the pen having index "PenIndex." Returns variant arrays with all the information, including authors. Supported only in animation runtime mode and for a history pen.
boolean InsertPenComment(long PenIndex, DATE CommentTDate, BSTR Author, boolean GroupComment, BSTR CommentText)	Inserts a historical comment for the historical pen with index "PenIndex." If "GroupComment" is set to TRUE, the inserted comment will be visible by all historical signals of the same data-logging group as the historical pen.
boolean ShowComments()	Toggles in animation (runtime) mode and batch information. If used, the Viewer will be placed in freeze mode, and all historical

	comments as well as batch information for the historical pens will be retrieved.
void SetPenLoAlarmString(BSTR PenName, BSTR AlarmString, BSTR PenType) **void SetPenLoLoAlarmString(BSTR PenName, BSTR AlarmString, BSTR PenType)** **void SetPenHiAlarmString(BSTR PenName, BSTR AlarmString, BSTR PenType)** **void SetPenHiHiAlarmString(BSTR PenName, BSTR AlarmString, BSTR PenType)**	The TrendWorX32 Viewer ActiveX allows the connection of OPC Data Access tags to its alarm limits. The OPC-enabled alarm limits will hold the most current value of the alarm limits. The alarm limits can be a constant value, an OPC Data Access tag, or an expression. In addition, the alarm limits are interfaced to the Global Alias and Language Alias servers. This functionality is not available in History Trend mode due to the nature of this specific historical replay mode. These OLE automation methods have been added to support this functionality.

ProjectWorX32 Support

Several OLE Automation methods have been added to facilitate ProjectWorX32 deployment and WebHMI enhanced support:

1. **ReplaceStringInString(**
   StringToReplace as String,
   OldSubstring as String,
   NewSubstring as String
) as String

2. **ReplaceHost(**
   OldHostName as String,
   NewHostName as String
) as Long

3. **ReplaceFilePath(**
   OldSubstring as String,
   NewSubstring as String
) as Long

4. **ReplaceTag(**
   OldSubstring as String,
   NewSubstring as String
) as Long

5. **ReplaceHostEx(**
   OldHostNameSubstring as String,

NewHostNameSubstring as String
MatchCase as Boolean,
MatchWholeWord as Boolean
) as Long

The **ReplaceHostEx** with parameters **MatchCase=False & MatchWholeWord=True** works exactly the same as the **ReplaceHost()** function.

Events Fired

ReadyStateChange	Fired when the Viewer has finished initializing its state.
InAnimateMode	Fired when the Viewer has entered automation mode.
UnableToCreateThread	Fired when the Viewer cannot create a background trend upon entering animation mode.
OutOfAnimateMode	Fired when the Viewer exits automation mode.
DatabaseChanged	Fired when there are configuration changes, such as adding or deleting a pen.
ErrorInLoadingData	Fired when the Viewer encounters errors upon loading its configuration data through a .v32 file.
RightTimeChanged	Fired when the "right" reference time of the Viewer changed while in freeze mode.
void CursorChangedPos()	Fired in freeze mode as the cursor changes position. You can call the "GetSampleInfo" method to retrieve detailed sample information on a per-pen basis.

Properties and Methods for Supporting Objects

Title Objects
Time Settings Objects
Ranges Objects
Pen Objects
Details Objects
Grids Objects

Title Objects

Title Objects: Properties
Title Objects: Methods
Using Visual Basic To View Title Objects

Title Objects: Properties

boolean ShowTitle	Determines whether the main title is displayed. Default: TRUE
BSTR TitleText	The main title of the Viewer ActiveX.

Title Objects: Methods

None

Time Settings Objects

Time Settings Objects: Properties
Time Settings Objects: Methods
Using Visual Basic To View Time Settings Objects

Time Settings Objects: Properties

boolean ShowDate	Determines whether date information is displayed in the Viewer's time field. Default: TRUE
boolean ShowMSecs	Shows milliseconds divisions on time axis. Default: FALSE
boolean ShowTimeInfo	Determines whether the time field is displayed. Default: TRUE
BSTR BorderEdge	Visual appearance of time display. Values are: sunken, raised, etched, and bumped. Default: Sunken
BSTR BorderOptions	Visual appearance options of time display. Values are: rectangle, left, bottom, right, top, top left, top right, bottom left, and bottom right. Default: Rectangle
BSTR DateFormat	Not used in the current version.
BSTR Size	Size for time display. It can be large, normal, or small. Default: Normal
long DataUpdatePeriod	This is the data-collection rate in milliseconds. "Set" is not supported in runtime (animation) mode.
long DataTotalPeriod	This is the trend period in seconds. "Set" is not supported in runtime (animation) mode.

long Samples	This is the number of samples per pen. "Set" is not supported in runtime (animation) mode.
long DataDisplayMult	Gets/sets drawing intervals relative to data collection.
OLE_COLOR FillColor	Color for filling time area.
OLE_COLOR TextColor	Color for text displayed.
long HistoryRefreshRate	Gets/sets the history refresh rate for which historical pens will be refreshed, regardless of the data-collection rate. Ideally, it should be configured to be as close as possible to the logging-to-disk interval set in the TrendWorX32 Configurator.
short TimeStampGap	Gets/sets the time stamp gap.
short MinorTick	Gets/sets the number of minor ticks in the time area.

Time Settings Objects: Methods

None

Ranges Objects

Ranges Objects: Properties
Ranges Objects: Methods
Using Visual Basic To View Ranges Objects

Ranges Objects: Properties

boolean OverHighRange	Gets/sets override for high range. If set TRUE, the high-range value will be computed from a tag or expression.
boolean OverLowRange	Gets/sets override for low range. If set TRUE, the low-range value will be computed from a tag or expression.
Short VisiblePens	Gets/sets the number of visible pen ranges on the ranges display area (if the **Global Ranges** option is not selected).
boolean ShowRanges	Determines whether the range field is displayed. Default: TRUE
BSTR BorderEdge	Visual appearance of range display. Values are: sunken, raised, etched, and bumped. Default: Sunken
BSTR BorderOptions	Visual appearance options of range display. Values are: rectangle, left, bottom, right, top, top left, top right, bottom left, and bottom right. Default: Rectangle
BSTR NumberFormat	Gets/sets number format for range values.
BSTR Position	Gets/sets position of range display. It can be right or left.
BSTR XNumberFormat	Gets/sets number format for *x*-axis range values in XY plots.
double XHigh	Gets/sets high-range value for *x*-axis in XY plots.

double XLow	Gets/sets low-range value for *x*-axis in XY plots.
double High	Gets/sets high-range value.
double Low	Gets/sets low-range value.
OLE_COLOR FillColor	Gets/sets fill color of range display.
OLE_COLOR TextColor	Gets/sets text color of range display.
BSTR Size	Size for time display. It can be large, normal, or small. Default: Normal
short MinorTick	Gets/sets the number of minor ticks in the range area.

Ranges Objects: Methods

None

Pen Objects

Pen Objects: Properties
Pen Objects: Methods
Using Visual Basic To View Pen Objects

Pen Objects: Properties

boolean Autoscale	Autoscales the pen.
boolean EnableAlarmLines	Enables drawing of alarm lines**.**
boolean OverHighRange	Gets/sets override for high range. If set TRUE, the high-range value will be computed from a tag or expression.
boolean OverLowRange	Gets/sets override for low range. If set TRUE, the low-range value will be computed from a tag or expression.
boolean PenVisible	Determines whether a pen is to be drawn.
boolean PenXAxis	Gets/sets a pen *x*-axis in XY-plots.
boolean ShowMarkers	If TRUE, a triangular marker is displayed at the position of the last known sample value.
boolean ShowSamples	Indicates real samples.
boolean Stairplot	Enables step-wise plotting mode.
boolean UpdateServerRanges	Controls the initial update of the ranges from the server.
BSTR DetailsFormat	The format type for the details.
BSTR PenPointName	The pen signal point name.
BSTR PenStyle	Gets/sets the pen style, (e.g. solid, dotted, dashed).
BSTR PenText	Pen description.

BSTR PenUnits	Pen engineering units.
BSTR RangeFormat	The format type for showing ranges.
BSTR TWXBgGroupName	The Persistent Trend group name.
double AlarmHigh	High alarm value for pen.
double AlarmHiHi	High-high alarm value for pen.
double AlarmLoLo	Low-low alarm value for pen**.**
double AlarmLow	Low alarm value for pen.
double HighLimit	High limit value.
double LowLimit	Low limit value.
long NumberOfSamples	Gets number of samples for the pen.
OLE_COLOR DetailColor	The fill color to be used in the **Details** window.
OLE_COLOR PenColor	Gets/sets the color associated with the pen.
OLE_COLOR RangeColor	The fill color to be used in the range window.
OLE_COLOR RangeTextColor	The color to be used for text in the range window.
short HistoryFilter	Returns the filter type for the historical data.
short PenWidth	The width of the pen in logical units. Default: 1
BSTR PenType	Returns the type of the pen, such as real-time, history, and custom. Read only property.
short HistoryFilter	Returns the history filter for the pen. Possible values are:

	Raw = 0 Average = 1 Minimum = 2 Maximum = 3 Standard deviation = 4 Last sample = 5 Minimum with actual time = 6 Maximum with actual time = 7
double AutoResetLevel	Gets/sets the autoreset value for a pen in configuration mode, if the totalizer mode is enabled.
double PreLoadValue	Gets/sets the preload value for a pen in configuration mode, if the totalizer mode is enabled.
double PresetValue	Gets/sets the preset value for a pen in configuration mode, if the totalizer mode is enabled.
double IntegrationFactor	Gets/sets the Integration factor value for a pen in configuration mode, if the totalizer mode is enabled. Possible values are: 1.0 (integration constant computed in seconds) 60.0 (integration constant computed in minutes) 3660.0 (integration constant computed in hours) 86400.0 (integration constant computed in days)
long IntegrationDir	Gets/sets the integration direction value for a pen in configuration mode, if the totalizer mode is enabled. Possible values are: 0 (Totalizer counts upward from 0.) 1 (Totalizer counts downward to 0.)
long TotalizerMode	Gets/sets the totalizer mode value for a pen in configuration mode if the totalizer mode is enabled. Possible values are:

	0 (continuous mode) 1 (preload mode) 2 (preset)
boolean EnableTotalizer	Enables/disables the totalizer mode for a pen in configuration mode.
long PlotterMark	Gets/sets the totalizer plotter mark size for the pen's curve, if the totalizer mode is enabled. Possible values are: No marker = 0 Tiny = 1 Small = 2 Medium = 3 Big = 4 Huge = 5
long TotalizerUnits	Gets/sets the totalizer units value for a pen in configuration mode, if the totalizer mode is enabled. Possible values are: 0 (integration factor computed in seconds; units are seconds) 1 (integration factor computed in minutes; units are minutes) 2 (integration factor computed in hours; units are hours) 3 (integration factor computed in days; units are days)
boolean IdealPenMode	Gets/sets the ideal pen mode for a historical pen.
boolean AutoTimeSync	Gets/sets the autosync mode for a real time pen.

Pen Objects: Methods

boolean GetPenStat(double* Max, double* Min, double* Avg, double* StdDev, long*	Returns the associated statistical values for the pen using the visible samples at the current state of the pen.

NumSamples)	
boolean IsHistoryPen()	Returns TRUE if the pen is historical.
boolean SetIdealDate(DATE Date)	Sets the ideal date for a historical ideal pen. This is the reference starting time of the ideal pen.
DATE GetIdealDate()	Sets the Ideal date for a historical ideal pen. This is the reference starting time of the ideal pen.

Details Objects

Details Objects: Properties
Details Objects: Methods
Using Visual Basic To View Details Objects

Details Objects: Properties

boolean ShowDetails	Determines whether the details field is displayed. Default: TRUE
boolean ShowPenDate	Determines whether a pen's date information is displayed.
boolean ShowPenHiHiLoLoAlarm	Determines whether a pen's "HiHi" and " "LoLo" alarm information is displayed.
boolean ShowPenHiLo	Determines whether a pen's "Hi/Lo" information is displayed.
boolean ShowPenHiLoAlarm	Determines whether a pen's "Hi/Lo" alarm information is displayed.
boolean ShowPenPoint	Determines whether a pen's information is displayed.
boolean ShowPenText	Determines whether a pen's text is displayed.
boolean ShowPenTime	Determines whether a pen's time information is displayed.
boolean ShowPenUnits	Determines whether a pen's units are displayed.
boolean ShowPenValidity	Determines whether signal quality will be displayed.
boolean ShowPenValue	Determines whether a pen's value is displayed.
boolean ShowTooltips	TRUE if plot ToolTips are to be shown for the currently focused pen (when in freeze mode).

boolean ShowXAxis	Determines whether *x*-axis values are shown for XY plots.
BSTR BorderEdge	Visual appearance of the **Details** window.
BSTR BorderOptions	Visual appearance of the **Details** window.
Short Visible Pens	Gets/sets the number of visible pen details to be drawn in the **Details** window.
BSTR NumberFormat	Gets/sets the number format of the pen.
BSTR Size	Gets/sets the size of the **Details** window.
OLE_COLOR FillColor	The background color of the **Details** window.
OLE_COLOR TextColor	The color of all text in the **Details** window.
boolean ShowHeader	Enables/disables the display of the header in the **Details** window.
boolean ShowFilter	Enables/disables the display of the data filter in the **Details** window.
boolean ShowMSecs	Enables/disables the display of milliseconds in the time column of the details window.
boolean ShowPenStyle	Enables/disables the display of the pen's style in the **Details** window, rather than the squares filled in the color of the pen.

Details Objects: Methods

None

Grid Objects

Grid Objects: Properties
Grid Objects: Methods
Using Visual Basic To View Grid Objects

Grid Objects: Properties

boolean ShowGrids	Determines whether grids are displayed in the Viewer's main display area. Default: TRUE.
BSTR GridStyle	Gets/sets grid style, such as solid, dotted, and dashed.
OLE_COLOR GridColor	The color of the grid.
short GridNumber	Gets/sets grid number
short GridWidth	The grid width in logical units.
short SubGridNumber	Gets/sets the subgrid number for the Viewer in configuration mode, if the circular chart mode is enabled.
short SubGridWidth	Gets/sets the subgrid width for the Viewer, if the circular chart mode is enabled.
BSTR SubGridStyle	Gets/sets the subgrid style for the Viewer, if the circular chart mode is enabled.
boolean ShowSubGrid	Enables/disables the display of the subgrids number for the Viewer, if the circular chart mode is enabled.
OLE_COLOR SubGridColor	Gets/sets the subgrid color the for Viewer, if the circular chart mode is enabled.
short GridsBetweenLabels	Gets/sets the grids between labels value in the Viewer, if the circular chart mode is enabled.
short CircularEdgeMode	Gets/sets the circular edge appearance for the Viewer, if the circular chart mode is enabled. Possible values are:

	0 (No edge) 1 (Default) 2 (Thick)

Grid Objects: Methods

None

Code Samples

Using the Component Inside Visual Basic

The TrendWorX32 Viewer ActiveX easily integrates into any Visual C++ or Visual Basic container application. The following are some code samples for using the TrendWorX32 Viewer ActiveX inside Visual Basic:

a. Disable Global Ranges and animate the component

```
With TWXViewer1
        .GlobalRanges = False
        .EnterAnimate
End With
```

b. Freeze the Trend

```
With TWXViewer1
        .FreezeMode = True
End With
```

c. Unfreeze the Trend

```
With TWXViewer1
        .FreezeMode = False
End With
```

d. Modify X- axis Grid Style

```
Dim xGrids As Object
'Get XGrids Interface
Set xGrids = TWXViewer1.xGrids
With xGrids
        .GridStyle = "Dotted"
        .GridNumber = 8
End With

' Call Refresh method
TWXViewer1.Refresh
'Release Interface
Set xGrids = Nothing
```

e. A full script

```
' Get Pen Names
Dim penNames As Variant
Dim Count As Integer
Dim penDel As String
penNames = TWXViewer1.GetPenNames
'Show to the user
For Count = 1 To UBound(penNames)
                penDel = penNames(Count)
                MsgBox (penDel)
Next Count
' Now disable pen drawing for last pen
Dim pen As Object
'Get Pen's Interface
Set pen = TWXViewer1.GetPen(penDel, "Real Time")
MsgBox ("Hiding Last Pen !")
'Hide Pen
With pen
```

```
                .PenVisible = False
            End With

        'Call refresh method
            TWXViewer1.Refresh
            MsgBox ("Showing Last Pen !")
            'Show pen again
            With pen
                .PenVisible = True
            End With

        'Call Refresh method
            TWXViewer1.Refresh
            MsgBox ("Deleting  Last Pen !")
            'Now delete this pen
            'Release Pen Interface
            Set pen = Nothing
            'Delete pen
            TWXViewer1.DeletePen penDel, "Real Time"
            'Call refresh method
            TWXViewer1.Refresh
```

f. Exit Runtime (Animate Mode)

```
        With TWXViewer1
            .ExitAnimate
        End With
```

Note: All of the OLE Automation objects exposed by the TrendWorX32 Viewer ActiveX are already created when the component is installed. The programmer should get the interfaces and not try to create them.

When you programmatically obtain an interface to one of the TrendWorX32 objects, the reference count for that interface is increased. Therefore, you should always release the corresponding interface once it is not needed anymore. In addition, you should *always* check for "null" (or "nothing" in Visual Basic) interface pointers. Depending on the state of the TrendWorX32 Viewer component and the configured security settings, you may not have access to a certain object, and the returned interface in this case is a "null" interface.

VBA Examples

The GENESIS32 installation includes a set of examples for the TrendWorX32 Viewer ActiveX. The examples are located in the **\Program Files\ICONICS\GENESIS32\Examples\TrendWorX32** directory, which contains the following folders:

Aliasing. The files in the "Aliasing" folder are example files that demonstrate the use of the OLE Automation interface of the TrendWorX32 Viewer ActiveX within a VBA-enabled application, such as GraphWorX32, to do aliasing.

Excel Templates. The "EXCEL Templates" folder contains example Microsoft Excel templates that can be used when creating Excel workbooks to be used with TrendWorX32 Reporting.

Real-Time Trend Configuration. The files in the "Real Time Trend Configuration" folder demonstrate the use of the OLE Automation interface of the TrendWorX32 Viewer ActiveX within a VBA-enabled application, such as

GraphWorX32. It demonstrates how to add pens "on the fly" by reading the pen configuration from a Microsoft Access database (.mdb) file and utilizing the OLE Automation properties of the TrendWorX32 ActiveX through VBA scripts.

TrendWorX32 Data Logging and Reporting. The "TrendWorX Data Logging and Reporting" folder contains example files that demonstrate the use of the TrendWorX32 Data Logger and Reporting modules to create reports to an existing, user-configured Excel Workbook. This data logging demo uses the default TrendWorX32 configuration file and data files installed with GENESIS32. The default historical data file, called **TwxLogdata.mdb**, is located under "\Documents and Settings\All Users\ICONICS\Examples32".

TrendWorX32 VBA Examples. The "TrendWorX32 VBA Examples" folder contains various examples that demonstrate the use of the OLE Automation interface of the various TrendWorX32 modules and components.

VB Script Examples. The "VB Script Examples" folder contains a **TWX-VBScriptEx.gdf** display that uses VB scripting. Open this file in GraphWorX32 and go into runtime mode, as shown in the figure below. Click on the various buttons to see how the VBScript functions work. In configuration mode, you can click on the buttons to see how the script pick actions are configured.