



Your comments and suggestions on the operation of this software are welcome. Please address them to:

ICONICS
100 Foxborough Blvd.
Foxborough, MA 02035
Tel: 508-543-8600
Fax: 508-543-1503
E-Mail: support@iconics.com
Web: www.iconics.com

© ICONICS, Inc. All rights reserved.

Contents

Chapter 1: Getting Started
Chapter 2: System Configuration
Chapter 3: Managing Display Files
Chapter 4: Creating and Modifying Objects
Chapter 5: Display Edit Functions
Chapter 6: Viewing Files
Chapter 7: Data Connections
Chapter 8: Dynamic Connections
Chapter 9: Tools
Chapter 10: Runtime Environment
Chapter 11: VBA Wizards
Chapter 12: GraphWorX32 ActiveX Controls
Chapter 13: OLE Automation Reference
Chapter 14: GraphWorX32 VBA Project

Chapter 1: Getting Started

Contents

Introduction to GraphWorX32
Starting GraphworX32
GraphWorX32 Screen
Using the Mouse and Keyboard

Introduction to GraphWorX32

GraphWorX32 is a human-machine interface (HMI) software package for process control. GraphWorX32 is a fully compliant OPC client featuring ActiveX® and OLE Automation technologies.

Key Features of GraphWorX32

- VCRWorX32 data replay.
- Data mining open database connections.
- "Run reports" pick action.
- "Run transactions" pick action.
- "Database updates" pick action.
- Format painter.
- Confirmation message builder.
- Unified Data Browser.
- Microsoft VBA support.
- Global alasing support.
- TraceWorX32 integration for debugging and diagnostics.
- Dockable Symbol Library with OLE Automation.
- Dockable VBScript and JScript editors.
- Web Publishing Wizard for exporting and publishing displays to HTML.
- Transparent to displays and objects.
- Display Thumbnail to File Open Function.
- Publish multiple graphics pages to Web server.
- Process point and text improvements.
- Screen updates for foreground and dynamics.
- Find and replace support for state fields.
- Find and replace support for local aliases.
- Simplified access to symbol tags and aliases.
- Enhanced Expression Editor.
- AlarmWorX32 Indicator ActiveX added to GraphWorX32 toolbar.
- VBScript examples in Symbol Library.
- Integrated database support.
- Support for language alias browsing.
- Support for global alias browsing.
- Expanded image file import and export functions.
- Image compression.
- Improved runtime zoom and pan.
- Automation improvements.
- Printing enhancements.
- Save as previous version support.
- International language-switching support.

- Global alaising in configuration and runtime dialogs.
- VBA runtime mouse and keyboard events.
- Digital selector, analog selector, and animator enhancements.
- Pick action enhancements for layers.
- Object selection enhancements.
- Event log enhancements for logging actions and data.
- Transparency and translucency.
- Zoom limits for box zoom.
- OLExpress 6.0 compatibility.
- GenStatistics runtime data statistics viewer.
- Multiply custom command support.
- Security enhancements.
- Free rotation of objects.
- Edit data source connection capability.
- ActiveSync download to Windows CE and Pocket PCs.
- Pocket PC development mode.
- Real-time thread priority capability for data thread and timer thread.
- Function key and keyboard assignment capability.
- User-defined pop-up menus.
- OPC request types.
- Range support for the number pad.
- Numerous user interface enhancements to improve ease-of-use.
- Key disable capability through the GENESIS32 Secure Desktop Configurator.
- Universal OLE for Process Control (OPC) connectivity.
- True 32-bit design.
- A fully compliant OPC client.
- Powerful display-creation tools.
- A complete set of drawing and animation tools in an object-oriented environment. Displays can be scaleable (display automatically resizes when you stretch the window) or fixed-scale (for pixel-perfect displays no matter what size the window is).
- Containment of ActiveX controls and OLE objects.
- Embedding ActiveX controls from ICONICS or third-party ActiveX controls and OLE objects directly into your displays.
- ActiveX document server. GraphWorX32 is an ActiveX document server, which means its displays can be run in applications such as Microsoft® Internet Explorer.
- Comprehensive OLE Automation interface.
- A powerful set of OLE Automation methods and properties for programmatically manipulating GraphWorX32 displays.
- Fast dynamic animation.
- Capability of 50-millisecond updates.
- GraphWorX32 ActiveX control. GraphWorX32 includes an ActiveX control (**GWXview32.OCX**) that is capable of running GraphWorX32

displays. This component has all the runtime capabilities of **GWX32.exe,** and can be conveniently embedded into ActiveX languages, such as Visual Basic or HTML pages.

- Advanced Symbol Library. GraphWorX32 includes a utility for loading, storing, and organizing GraphWorX32 symbols. Simply drag-and-drop symbols to and from the Symbol Library.
- OPC Data Access compliant.
- More than 70 animation types supported.
- Development and configuration for Windows CE.
- Powerful AutoCAD style display layers with clutter/declutter.
- Gradient fills.
- State fields that display readable text representing machine states.
- Ability to rotate text in 90-degree increments.
- Web-based look and feel similar to Microsoft Internet Explorer.
- Professionally drawn 2-D and 3-D gradient symbol libraries.
- Runtime Window Properties Mode for improved WYSIWYG configuration.
- View toolbar for convenient access to zooming and layering commands.
- Powerful advanced polyline editing capabilities.
- Aliasing and advanced editing features.
- Password on display files and user-created custom symbols and symbol wizards.
- Smart ActiveX toolbar that allows you to add or remove any ActiveX control to the ActiveX toolbar.
- Keypad and QWERTY Keyboard support to process point dynamic.
- Customizable toolbars.

VBA Smart Wizard Symbols

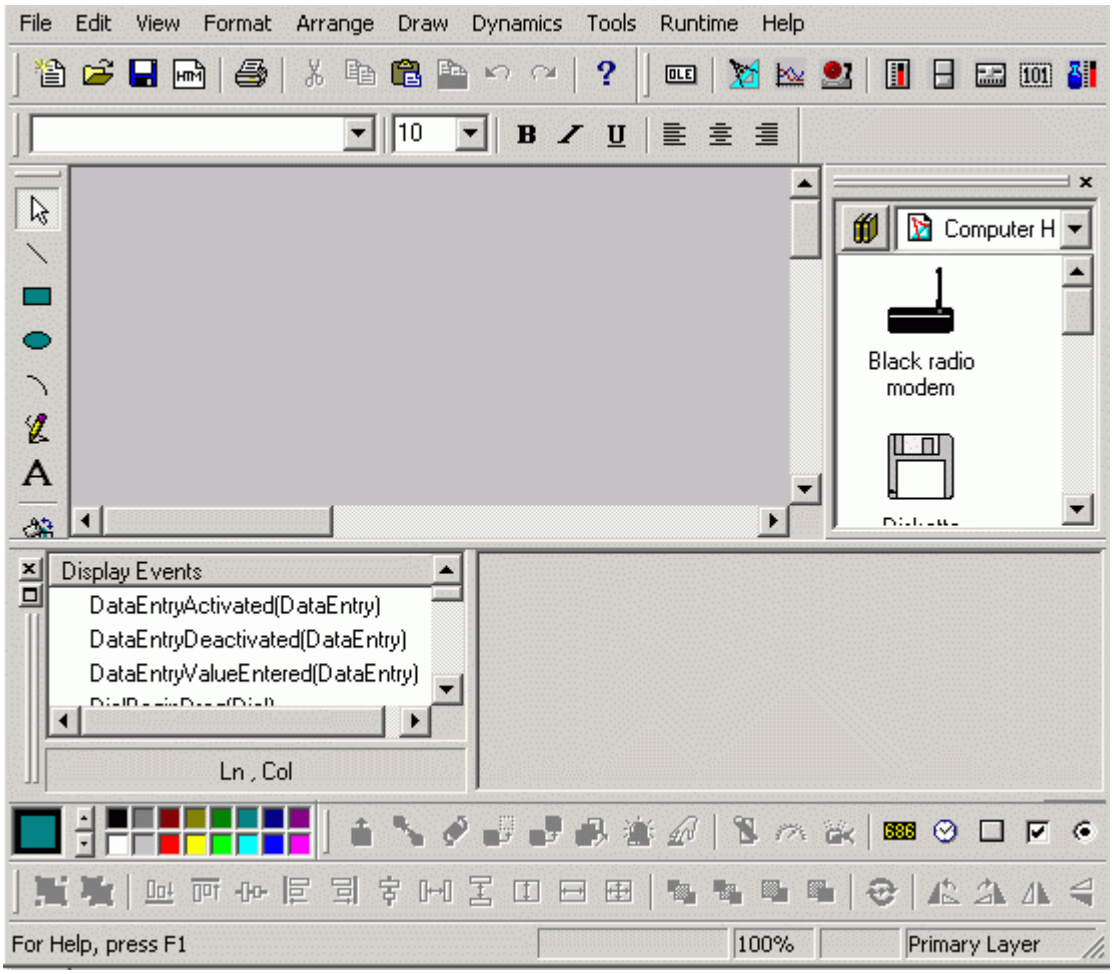
- **Help Wizard.** Provides detailed information on the use of VBA wizards.
- **OPC Data Access Wizard.** Sets and retrieves OPC data.
- **Microsoft Excel Wizard.** Integrates data to and from Excel.
- **Report Wizard.** Creates simple, useful reports.
- **Recipe Wizard.** Creates recipes.
- **Microsoft Word Wizard.** Integrates data into Microsoft Word.
- **E-mail and Outlook Wizard.** Sends e-mail based on VBA events.
- **OPC Calculator Wizard.** Connects a four-function calculator to a tag on any display.
- **OPC KeyPad Wizard.** Connects a QUERTY keypad to a tag on any display.
- **Runtime Aliasing Wizard.** Changes aliases during runtime mode.
- **Analog Chart Wizard.** Creates displays with analog OPC tags.
- **Digital Chart Wizard.** Creates displays with discrete OPC tags.

Starting GraphWorX32

To start GraphWorX32 from the Windows **Start** menu, select **Programs > ICONICS GENESIS32 > GraphWorX32 > GraphWorX32.** This opens the **gwx32.exe** file.

GraphWorX32 Screen

The figure below shows the GraphWorX32 screen with its basic components, including the Color Palette, Main toolbar, Draw toolbar, Arrange toolbar, Font toolbar, and Dynamics toolbar.



GraphWorX32 Screen

This section describes the features of the GraphWorX32 screen, as shown above. It also describes the tools used to create displays and make dynamic connections to data values. Refer to the Microsoft Windows documentation for general procedures regarding the Windows Graphical User Interface (GUI) and information about program groups, menus, dialog boxes, as well as file name extensions specific to windows.

GraphWorX32 Work Area

The **Work Area** is where you build your GraphWorX32 displays. It shows only a portion of the entire work area that can be accessed by the scroll bars or the **Zoom** function.

When a display is saved, information about the current view is saved with the display, so that it can be restored when the display is reloaded at a later time.

Definitions

Display size (or page size): Defines the world coordinate boundaries for the display. You can specify the display size.

Window dimensions: Refers to the size/location of the GraphWorX32 mainframe window.

Scaleable display: When the GraphWorX32 main window is resized, the view is scaled so that all objects currently in the view area remain visible.

Preserve aspect ratio: The ratio of object sizes in the display is maintained during window resizing.

Fixed-scale display: The view does not scale when the GraphWorX32 main window is resized. Visible objects are clipped or revealed if the window is sized smaller or larger.

View area: Area of the display that is currently visible.

Home view: View in which the display was last saved.

100% zoom: One logical unit equals one pixel. This is always the case, regardless of whether the display is fixed-scale or scaleable.

Status Bar

The status bar is displayed at the bottom of the GraphWorX32 window. To show or hide the status bar, select **Status Bar** from the **View** menu. The left side of the status bar describes the functions of menu items and toolbar buttons as you navigate through menus and toolbars.

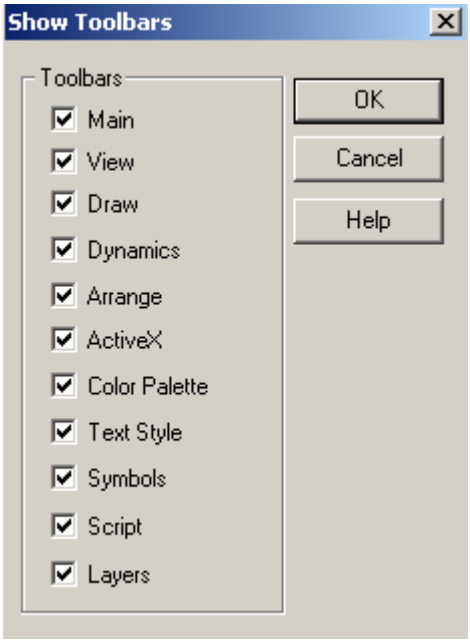
The following items are also shown in the status bar:

- Cursor coordinates
- Zoom percentage
- Grid snap on/off
- Display layers

Toolbars

The GraphWorX32 tools you use to create your displays are grouped by functionality and are available through the menu bar options and the following toolbars.

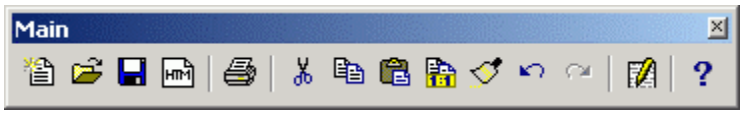
Selecting **Toolbars** from the **View** menu opens the **Show Toolbars** dialog box, shown below, which allows you to select which toolbars you want to be visible. The **Toggle Toolbars** command on the **View** menu hides or shows all of the toolbars at once.



Show Toolbars Dialog

Main Toolbar

The **Main** toolbar, shown below, is displayed by default across the top of the application window, below the menu bar. The toolbar provides quick mouse access to many tools used in your application. Use the **Main** toolbar for basic file and display maintenance such as creating a new display, opening a file, printing a file, and cutting, pasting and duplicating objects. It also allows you to undo and redo your last editing action, and to access help on your GraphWorX32 application.



Main Toolbar

View Toolbar

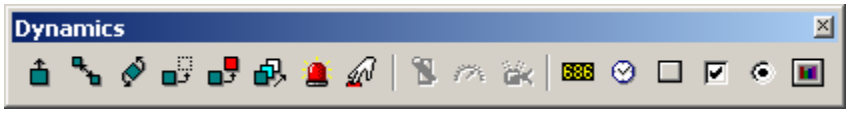
Using the **View** toolbar, shown below, you can change the view of the display by activating different zoom functions. Additionally, the toolbar provides specific shortcuts for layering. All of the following functions can also be found by selecting **Layers** from the **Format** menu.



View Toolbar

Dynamics Toolbar

You can use the **Dynamics** toolbar, shown below, to make dynamic connections to data points in OPC servers. See the **Dynamic Connections** section for a detailed description of functions on the **Dynamics** toolbar.



Dynamics Toolbar

ActiveX Toolbar

The **ActiveX** toolbar, shown below, enables you to insert various ActiveX objects into the GraphWorX32 display. Click the **OLE** button to browse for a list of objects.



ActiveX Toolbar

Draw Toolbar

You can use the **Draw** toolbar, shown below, to draw static graphic objects; to select objects; to fill or unfill objects, including drawing objects and text objects; and to import bitmaps, metafiles, and GraphWorX32 symbols.



Draw Toolbar

Arrange Toolbar

The **Arrange** toolbar, shown below, is used for grouping and ungrouping objects, aligning objects, rotating objects, and shifting objects in layers.

Note: It is possible to rotate text in a GraphWorX32 display. Text can only be rotated in 90-degree increments by using the **Rotate Left** and **Rotate Right** buttons. This ability to rotate text applies to:

- Multiline text
- Left, center and right alignments
- Text objects
- Process points
- Data entries
- Statefields
- Timedate
- Buttons

Check boxes and radio buttons cannot be rotated.



Arrange Toolbar

Text Style Toolbar

The **Text Style** toolbar, shown below, is used to change the font attributes of text objects in a display.



Text Style Toolbar

Layers Toolbar

Using the **Layers** toolbar, shown below, you can add, remove, or duplicate a layer. You can also edit the layer properties, set the active layer, set the next layer, set the previous layer, hide layers above the current layer and hide layers below the current layer. For more information, please see the **Layers** section.



Layers Toolbar

Note: The functions of each toolbar button are described in detail in later sections.

Dockable Symbol Toolbar

The Symbol Library is available in GraphWorX32 as a dockable toolbar or a floating window inside display files. It can be freely floating above a GraphWorX32 application or docked to any side of the GraphWorX32 display: left, top, right, or bottom, as shown in the figure below. Features of the Symbol Library are fully implemented in the Dockable Symbol Toolbar. You can create, rename, and delete both category (.sdf) files and symbols within the dockable toolbar, as well as unlock categories with passwords and change the look of the symbol icons in the view pane. The category (.sdf) file format of the Dockable Symbol Toolbar is fully compatible with previous versions of the Symbol Library, and vice versa.

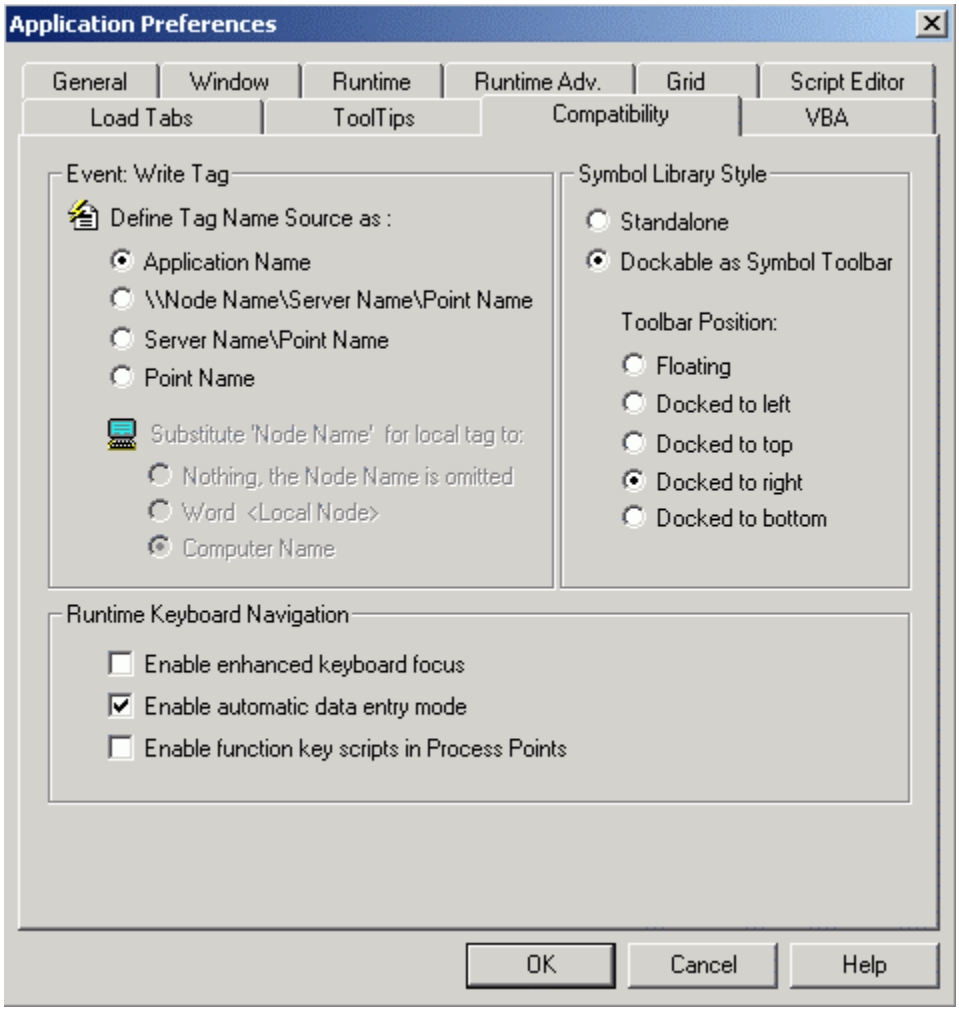
GraphWorX32 uses two different modes for the Symbol Library:

- Standard stand-alone Symbol Library
- Dockable symbol toolbar

To switch between these two modes:

1. Select **Application Preferences** from the **Format** menu.

2. The **Application Preferences** dialog box appears, as shown in the figure below. Select the **Compatibility** tab. Under the **Symbol Library Style** field, you can select **Stand-alone** mode or **Dockable as Symbol Toolbar** mode.

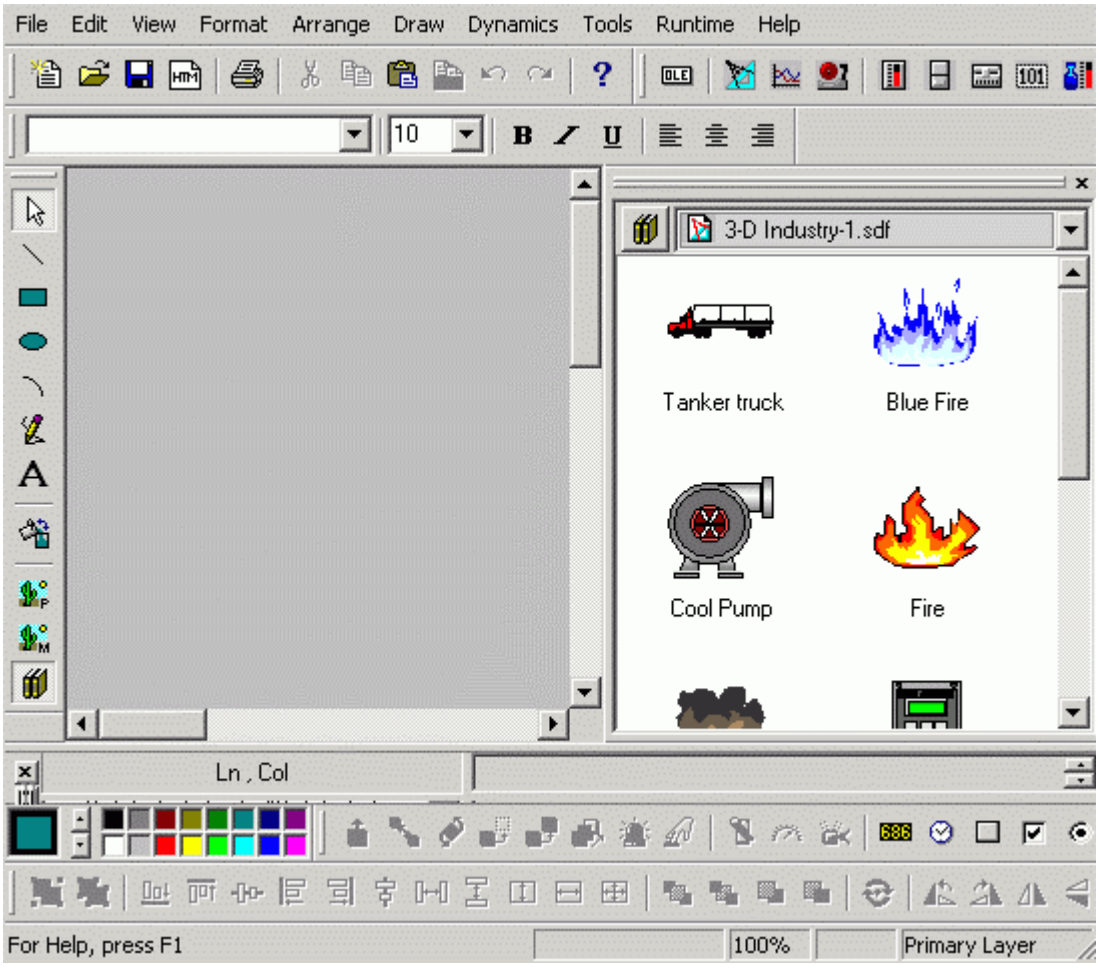


Switching Between Stand-alone and Dockable Symbol Modes

If **Stand-alone** mode is active, clicking the **Symbols** button on the **Draw** toolbar launches the **SymbolLibrary.exe.** You can also select **Import > Symbol** from the **Draw** menu.

If the **Dockable as Symbol Toolbar** option is selected, then the Symbol Library is replaced with a Symbol toolbar inside the GraphWorX32 display, as shown in the figure below. You can also position the symbol toolbar as:

- A floating window
- Docked to left
- Docked to right
- Docked to bottom
- Docked to top



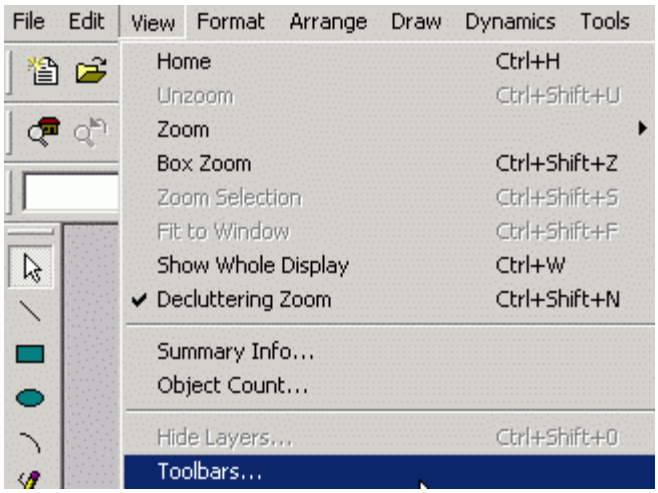
Symbol Library Docked Inside a GraphWorX32 Display

The **Symbols** button on the **Draw** toolbar toggles the visibility of the symbol toolbar (instead of launching the Symbol Library). The **Toggle Symbol Toolbar** command on the **View** menu is enabled, and the **Show Toolbars** dialog (which appears when you select **Toolbars** from the **View** menu) also includes a **Symbols** check box for showing/hiding the symbol toolbar. The **Import > Symbol** command is removed from the **Draw** menu.

Viewing the Dockable Symbol Toolbar in GraphWorX32

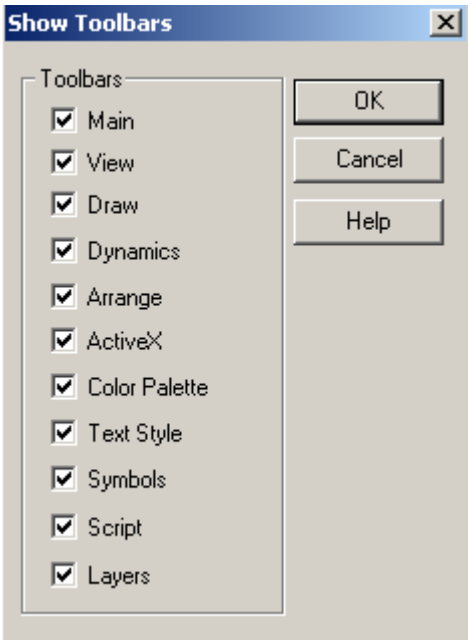
To view the Dockable Symbol Toolbar inside a GraphWorX32 display:

1. Select **Toolbars** from the GraphWorX32 **View** menu, as shown in the figure below.



Selecting Toolbars in GraphWorX32

2. In the **Show Toolbars** dialog box, check **Symbols,** as shown in the figure below. Click **OK.**

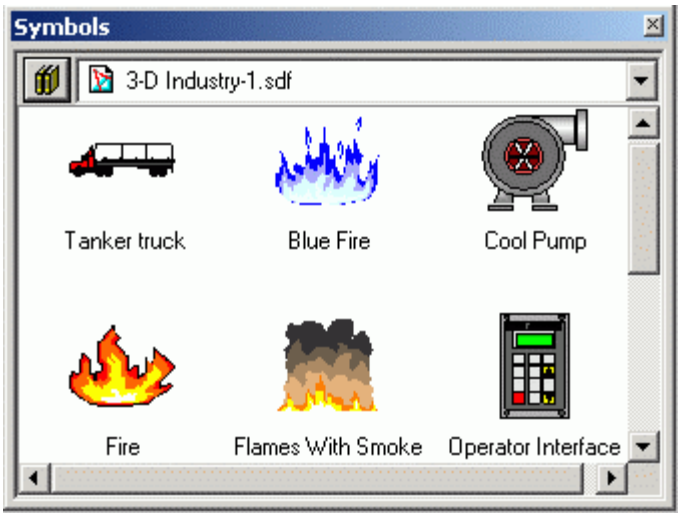


Selecting the Symbols Dockable Toolbar

3. This displays the Dockable Symbol Toolbar in the GraphWorX32 display.

Features of the Dockable Symbol Toolbar

The figure below shows the Dockable Symbol Library as a floating window. The drop-down list at the top of the window allows you to select from all available category (.sdf) files. GraphWorX32 symbols that are contained in the selected category file are displayed in the list control pane, as shown in the figure below. For maintenance and manipulation of symbols and categories, two pop-up menus are available.

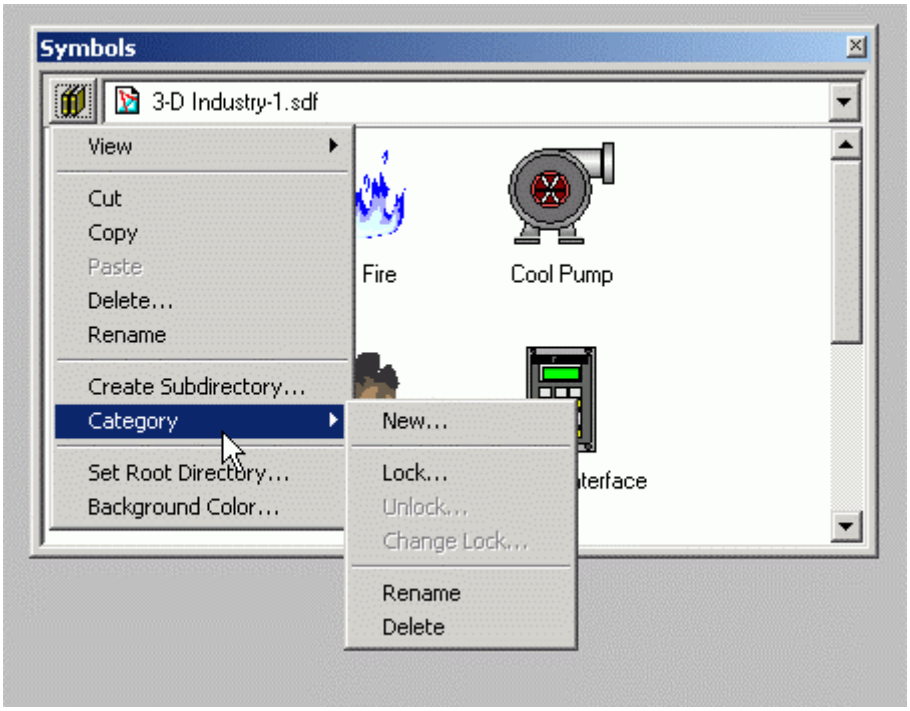


Dockable Symbol Library as a Floating Window

Symbol Toolbar Menu

The main pop-up menu for the Docking Symbol Toolbar is displayed when upper-left button is clicked, as shown in the figure below. You can also right-click anywhere inside the window to access this menu. This pop-up menu allows you to add, rename, delete, lock and unlock category files, as well as change the view of the symbol icons in the view pane. You can also cut, copy, paste, delete, and rename the symbols. The main pop-up menu also allows you to specify the root directory for the .sdf files. Previous versions of the Symbol Library maintained a tree control hierarchy of category files; in other words, category files could be placed under different subdirectories. The Dockable Symbol Library lists all category files as upper-level categories (even those included in subdirectories) and tracks where particular a category file came from so that no category files are lost.

Note: Please see the Symbol Library Help for complete information about menu functions.



Main Pop-up Menu in Dockable Symbol Toolbar

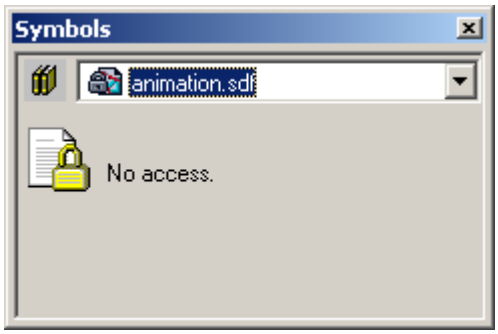
Symbol Toolbar Main Pop-up Menu Commands

Command	Function
View	Specifies the size of the symbol icons in the selected category.
Cut	Removes the selected symbol from the list view.
Copy	Copies the selected symbol.
Paste	Pastes a symbol into the list view.
Delete	Deletes the selected symbol.
Rename	Renames the selected symbol.
Create Subdirectory	Creates a new subdirectory (folder) in the root directory for the symbol category (.sdf) files.
Category	Adds, renames, deletes, locks, and unlocks symbol category files.
New	Creates a new category file.
Lock	Sets up the read-write password in the **Set Category Write Mode Password** dialog box. The password must be confirmed. Sets the read-write password to the category, which currently has no locks set, or the read-only password for the category, which is already opened in read-write mode. You cannot set the read-only password without setting the read-write password; this would result in a category file lock that no one would be

	able to remove (to modify the category file).
Unlock	Opens a locked symbol category. When the category file is initially locked, it is always locked with read-write access denied. (It could also have the read-only lock set). Simply enter the read-write password (or the read-only password if it is configured) to open the file. When the password is entered, it is recognized as either read-write or read-only. If the password is accepted, the category file transitions to the proper state to reflect the change.
Change Lock	Changes the category file password. You must be granted read-write permission to open the category with the read-write password.
Set Root Directory	Specifies the target directory for the symbol category (.sdf) files.
Rename	Renames the selected category file.
Delete	Deletes the selected category.
Background Color	Changes the background color of the list view pane.

Category Locks

It is possible to lock any category file included in the drop-down list for writing to and even for reading from the symbol category. If particular category is locked, it is indicated by different category icon in the list box, as shown in the figure below. Also, if the category file is locked both for reading and writing, a message will appear on the Dockable Symbol Toolbar notifying the user about the lock.



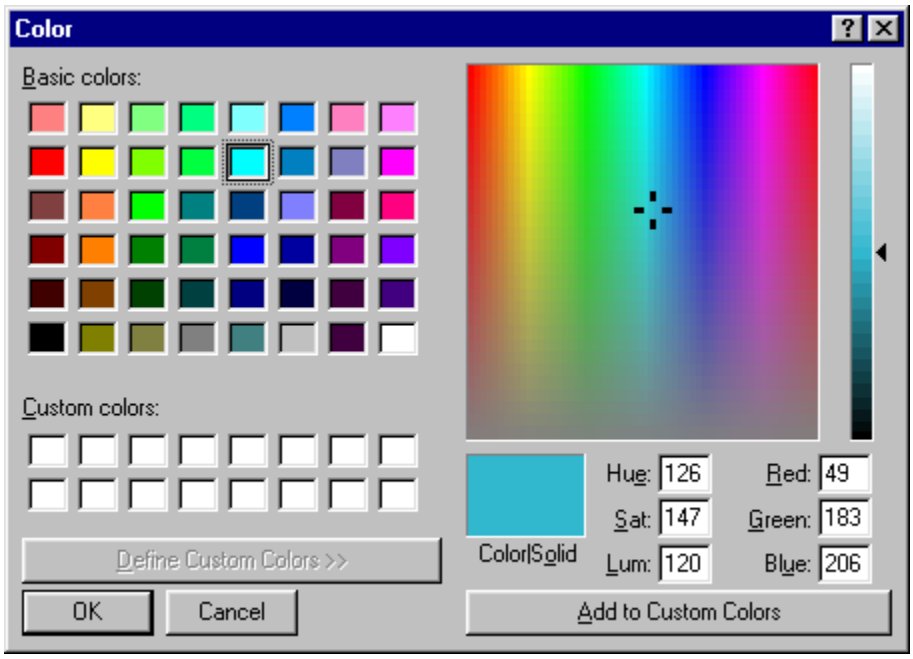
Locked Symbol Category File Indicated

Color Palette

A **Color Palette** tool is provided for selecting new colors during configuration mode in the GraphWorX32 application. This includes fill, border, background and text colors.The Color Palette, shown below, operates both as a toolbar and/or a dialog box.



Color Palette Toolbar



Color Palette Dialog Box

The color palette tool consists of two or more rows of selectable colors, a spin button control for accessing additional colors not currently visible, and a current selected color indicator.

Fill color is changed by left-clicking on one of the color boxes.

Line color is changed by right-clicking on one of the color boxes. For text objects, right-clicking on one of the color boxes changes the color of the text.

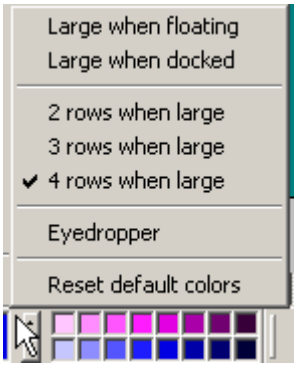
Double-clicking on a color box is used to select a custom color (via the windows custom color common dialog).

If two or more objects are grouped together, it is possible to change the fill and line color of both objects.

The toolbar can be reoriented while it is floating to switch between horizontal and vertical orientations.

Other Features of the Color Palette Tool

Right-clicking on the empty area surrounding the spin button control of the color palette toolbar opens a pop-up menu, as shown in the figure below.



Color Palette Tool Menu

This menu can be used to change the size of the color palette (large/small) and to set the number of rows of colors visible at a time (two, three, or four). There is also an option to reset the factory default color definitions of the color palette.

Note: The eyedropper feature can be used to pick up colors from a display to add them to the color palette tool. To use this, choose **Eyedropper** from the pop-up menu, as shown above (the mouse pointer will change to an eyedropper). Click anywhere in the GraphWorX32 window over the color you want to pick up. Deposit the color in the color palette by clicking over the desired color box in the rows of available colors.

Using the Mouse and Keyboard

You can use both the mouse and keyboard to control GraphWorX32. This section explains how to operate the mouse and lists keyboard shortcuts for GraphWorX32.

Mouse Functions

In Windows, you use the mouse to move a pointer on the screen, usually in the shape of an arrow. The pointer shows you where you are on the screen and allows you to point to and select application items.

Function	Description
Left-click	Selects static objects, such as rectangles and ellipses, tools, and menus.
Double-click	Double-clicking an object opens the Property Inspector.
SHIFT+Left-click	Selects multiple objects one by one.
Right-click	Opens the **Format** menu. Right-clicking on the empty display also shows some of the **Format** menu commands.
Drag (left mouse button) and object or an object's handles	Moves, stretches, and resizes objects. Docks and undocks toolbars.
Hold down SHIFT while stretching object	Keeps the sizes of objects in proportion after they are drawn.
Hold down CTRL while stretching object	Resizes from the center of an object.
Hold with CTRL + SHIFT	Stretches the object in proportion from the center.
Hold down ALT while grid is on	Object will not snap to grid and will turn off snap to grid temporarily.
Hold down SHIFT while moving object if move icon will not work	Will keep object in the same horizontal and vertical plane.
Drag-and-drop	Objects can be dragged and

	dropped to other instances of GraphWorX32 and the Symbol Library.
Drag-and-drop from GraphWorX32 display to desktop	Becomes an icon and can be dragged back as the original object.
CTRL+Drag	Copy and duplicate within the display.
Hold down SHIFT while creating a rectangle or a circle	Forces the object to be a perfect square or circle.
Hold down SHIFT while drawing a line	Line segments are drawn in increments of 45 degrees.
SHIFT+Right-click	To edit text, arc, line, or symbol objects.

Note: If objects are selected, use cursor keys to move the objects by one pixel.

Drag-and-Drop

GraphWorX32 supports the drag-and-drop of native GraphWorX32 objects and OLE objects. Static and dynamic GraphWorX32 objects can be dragged from one instance of GraphWorX32 into another instance of GraphWorX32 as native objects. Furthermore, objects can be dragged onto the Windows desktop (as a scrap) and can later be dropped back into GraphWorX32 as native objects. Objects dropped into GraphWorX32 that are not in the native GraphWorX32 format (or are not one of the other clipboard formats supported by GraphWorX32) become OLE embedded objects. Other supported clipboard formats include text, bitmaps, and Windows metafiles.

Keyboard Shortcuts

File Functions

CTRL+N	Opens a new file
CTRL +O	Opens an existing file
CTRL+S	Saves the current file
CTRL+P	Prints the current file

Edit Functions

CTRL+Z	Undo
CTRL+X	Cut

CTRL+C	Copy
CTRL+V	Paste
CTRL+D	Duplicate
DEL	Delete
CTRL+F	Find
CTRL+E	Replace
CTRL+A	Select all objects

View Functions

CTRL+H	Home
CTRL+SHIFT+U	Zoom out
CTRL+SHIFT+Z	Box zoom
CTRL+SHIFT+S	Zoom in
CTRL+SHIFT+F	Fit to window
CTRL+W	Show whole display
CTRL+SHIFT+N	Decluttering zoom
CTRL+SHIFT+O	Hides layers
CTRL+T	Toggles toolbars
CTRL+ALT+J	Toggle Script Toolbar
CTRL+ALT+S	Toggle Symbol Toolbar
CTRL+B	Shows/hides the status bar
CTRL+L	Toggles the scroll bars
CTRL+SHIFT+G	Shows the grid
CTRL+R	Runtime window properties mode
F4	Opens the **Properties** window
CTRL+ALT+U	Select Language
CTRL+0	Custom zoom
CTRL+1	Zoom 50%
CTRL+2	Zoom 75%
CTRL+3	Zoom 100%
CTRL+4	Zoom 150%
CTRL+5	Zoom 250%
CTRL+NUMPAD+	Zoom in
CTRL+NUMPAD-	Zoom out
CTRL+M	Toggles configuration and runtime modes

Arrange Functions

CTRL+G	Group into symbol
CTRL+U	Ungroup symbol
CTRL+SHIFT+PgUp	Brings object to front
CTRL+SHIFT+PgDn	Sends object to back
CTRL+PgUp	Brings object forward
CTRL+PgDn	Sends object backward

Align

CTRL+SHIFT+T	Tops
CTRL+SHIFT+B	Bottoms
CTRL+SHIFT+M	Middles
CTRL+SHIFT+L	Lefts
CTRL+SHIFT+R	Rights
CTRL+SHIFT+C	Centers

Space Evenly

CTRL+SHIFT+A	Across
CTRL+SHIFT+D	Down

Make Same Size

CTRL+SHIFT+H	Height
CTRL+SHIFT+W	Width
CTRL+SHIFT+O	Both

Move Object

ARROW LEFT	Moves the object 1 pixel to the left
ARROW RIGHT	Moves the object 1 pixel to the right
ARROW UP	Moves the object 1 pixel up
ARROW DOWN	Moves the object 1 pixel down
CTRL+ARROW LEFT	Moves the object 10 pixels to the left
CTRL+ARROW RIGHT	Moves the object 10 pixels to the right
CTRL+ARROW UP	Moves the object 10 pixels up
CTRL+ARROW DOWN	Moves the object 10 pixels down

Move Border

SHIFT+ARROW LEFT	Moves the right border 1 pixel to the left
SHIFT+ARROW RIGHT	Moves the right border 1 pixel to the right
SHIFT+ARROW UP	Moves the upper border 1 pixel up
SHIFT+ARROW DOWN	Moves the upper border 1 pixel down
CTRL+SHIFT+ARROW LEFT	Moves the left border 1 pixel to the left

CTRL+SHIFT+ARROW RIGHT	Moves the left border 1 pixel to the right
CTRL+SHIFT+ARROW UP	Moves the bottom border 1 pixel up
CTRL+SHIFT+ARROW DOWN	Moves the bottom border 1 pixel down

Chapter 2: System Configuration

Contents

Introduction
Command Line Options
Application Preferences and Display Properties
Copying and Resetting Preferences and Properties

Introduction

This section explains the different ways of configuring and customizing your system. You can customize your GraphWorX32 system using:

- Command line options.
- **Application Preferences** dialog box.
- **Display Properties** dialog box.

Command Line Options

You can customize a GraphWorX32 program item in Windows NT. This command line feature allows you to start GraphWorX32 with a specified display and launch GraphWorX32 directly in runtime.

GraphWorX32 supports the following options on the command line:

GWX32 [Displayname] [Options]

The command line options are listed in the table below.

Command Line Options

Option	Description
Runtime	Launches GraphWorX32 in runtime, automatically suppressing splash screen.
NoSplash	No splash screen.
RegServer	Registers application.
UnregServer	Unregisters application.
Themes	Global alias themes.
AliasFile	File name.
Position	x,y,w,h

Command Line Examples

In the following Command Line example, GraphWorX32 starts with the Tanks.gdf display file:

C:\Genesis-32\GWX32.EXE C: \Genesis-32\PROJECT1\TANKS.GDF

You can also combine two or more of these options. In the following example, GraphWorX32 starts in runtime with the tanks.gdf display loaded and the application window in the middle of your screen:

C:\Genesis-32\GWX32.EXE C: \Genesis-32\PROJECT1\TANKS.GDF -Runtime

This means that window size and position are already supported. Display size is not supported, and it is recommended that you use a simple VB script for advanced options that are currently not supported by the command line. The VB code is as simple as the following sample, and someone even could create a simple VB launcher, which itself will support a command line and transform it to automation calls.

The following VB/VBScript script will create an instance of GraphWorX32, start runtime, set the window size and position, and display it.

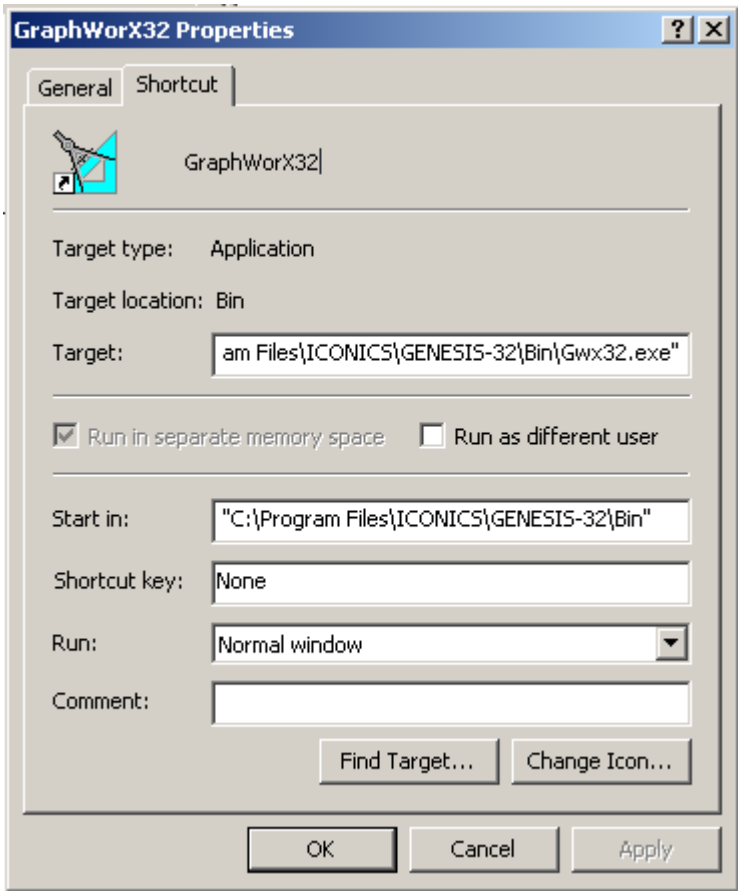
```
Dim g
Set g = CreateObject("GWX32.Display")
Call g.StartRuntime
Call g.SetWindowDimensionsPixels(0, 0, 600, 400)
Call g.ShowWindow
Call g.BringWindowToTop
Unload Me
```

Similarly there can be other commands added to load a desired display, specify the display size or window size and position as a percentage, and even preprocess the loaded display before showing it. This is much more powerful than any command line, and not much more difficult. But the command line supports the basic options too, so use them as desired.

Customizing the Launch of GraphWorX32 and Windows

To customize the launch of GraphWorX32 and Windows, use the following procedure:

1. From the Windows Start menu, select **Programs > ICONICS GENESIS32 > GraphWorX32 > GraphWorX32.**
2. Right-click on the **GraphWorX32** shortcut and select **Properties** from the pop-up menu.
3. The **GraphWorX32 Properties** dialog box opens, as shown below.
4. Click the **Shortcut** tab and enter the appropriate Command Line options at the end of the **Target** field.

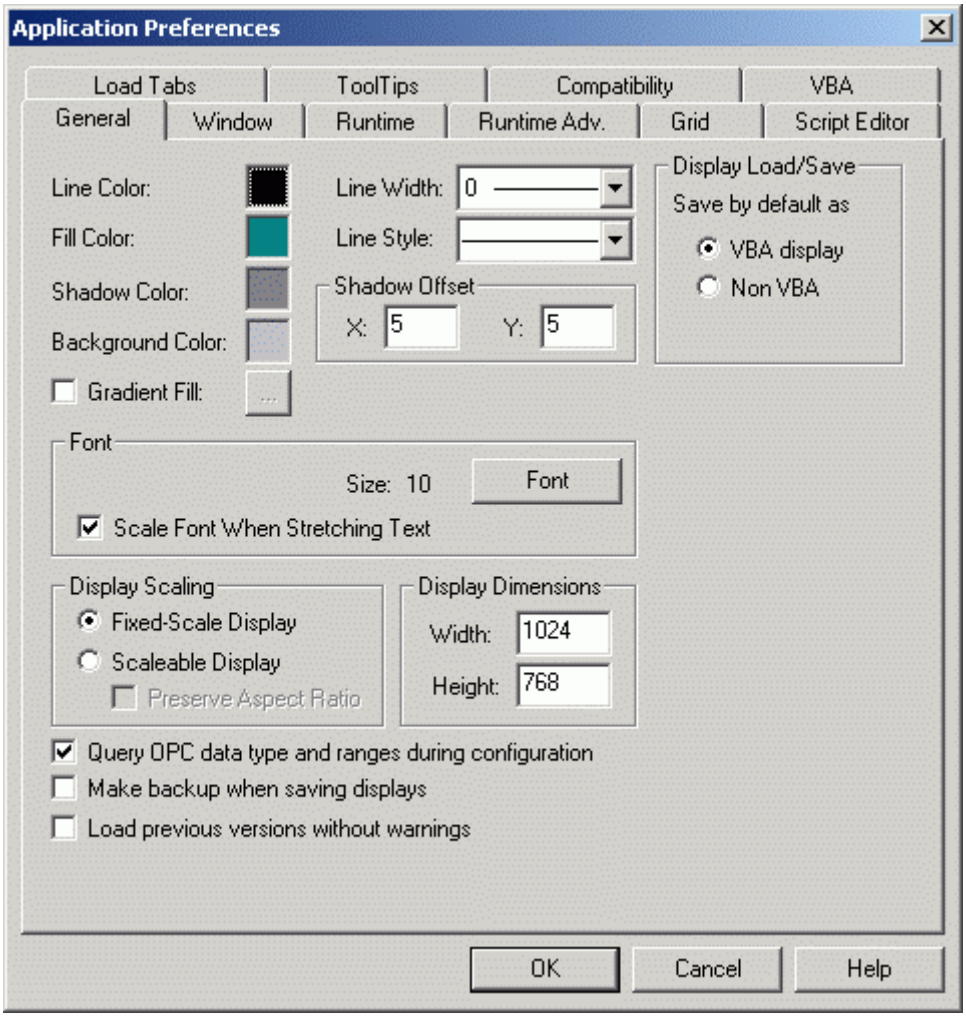


Shortcut to GraphWorX32 Properties Dialog Box

Application Preferences and Display Properties

Application Preferences

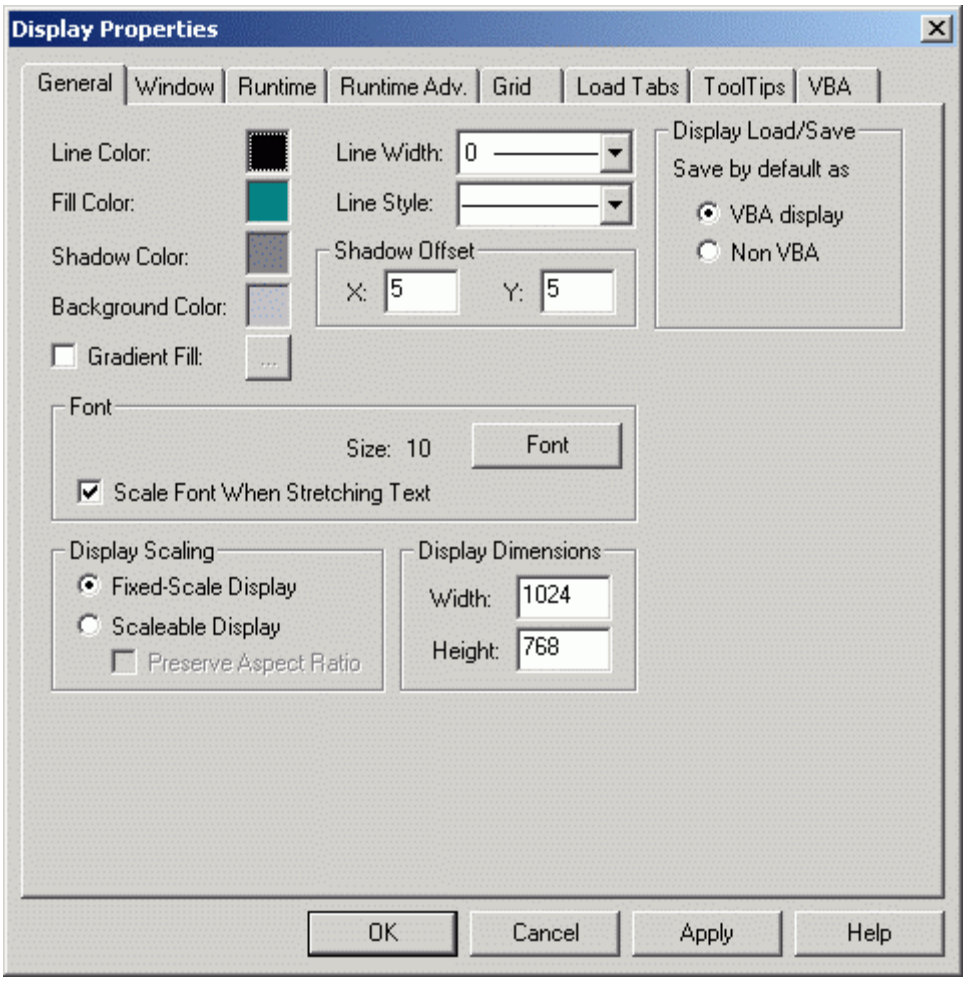
The **Application Preferences** dialog box, shown below, sets default operating parameters for your display-editing environment. These parameters apply to each new display created in GraphWorX32. Select **Application Preferences** from the **Format** menu. The **Application Preferences** dialog box opens, as shown in the figure below. Here you can set the defaults for the display. Since GraphWorX32 Preferences apply only to new displays, changes made in this dialog box will not be seen until you select **New** from the **File** menu.



Application Preferences Dialog Box

Display Properties

You can also define the properties for the current display by selecting **Display Properties** from the **Format** menu. This opens the **Display Properties** dialog box, shown below. Any changes you make in this dialog box will be immediately reflected in the display.



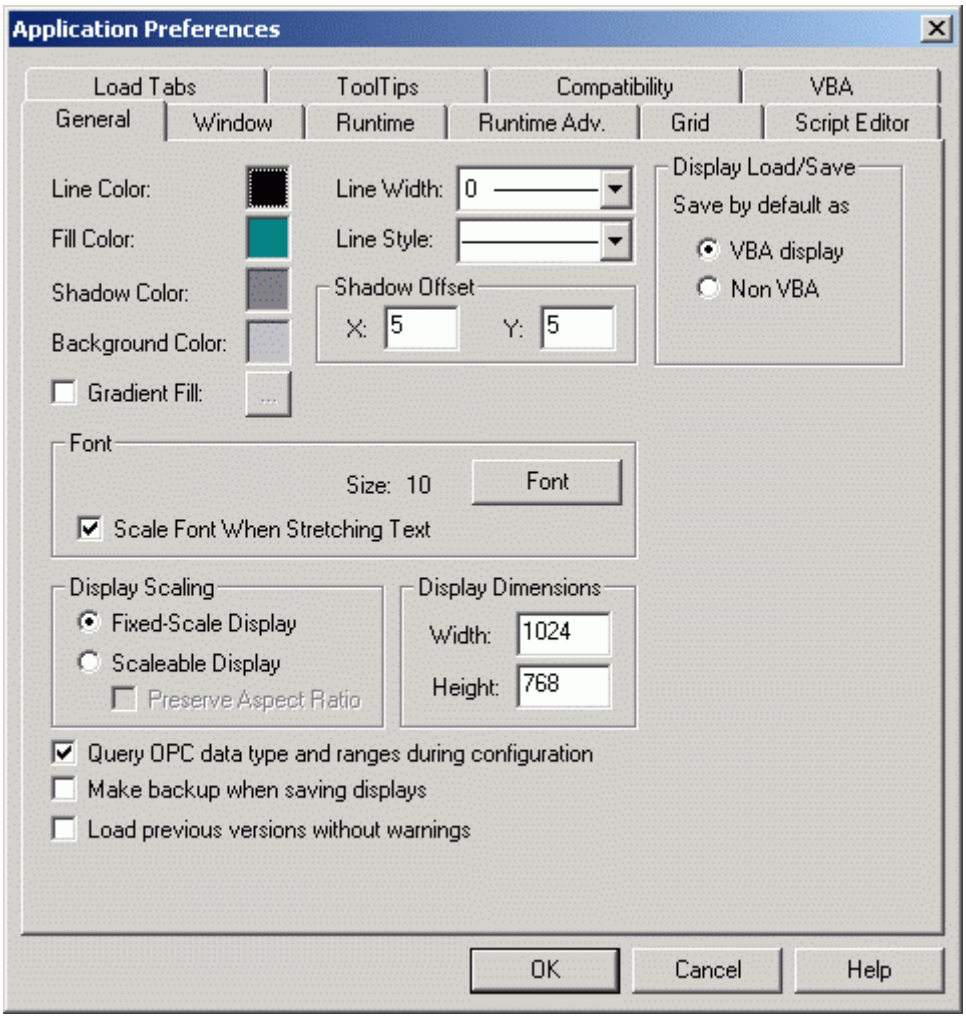
Display Properties Dialog Box

As you can see, the **Display Properties** dialog box is very similar to the **Application Preferences** dialog box. These dialog boxes contain the following tabs:

- General
- Window
- Runtime
- Runtime Advanced
- Grid
- Script Editor
- Load Tabs
- ToolTips
- Compatibility
- VBA

General Parameters

The **General** tab of the **Application Preferences** and **Display Properties** dialog boxes, shown below, adjusts the settings for the GraphWorX display.



Application Preferences Dialog Box: General Tab

Besides **Line Color** and **Fill Color,** which allow you to choose the color to use for defining lines and filling objects, **Background Color** provides choices of color for the display background. **Shadow Color** and **Shadow Offset** change the depth of a selected object. You can also choose the **Line Width** and **Line Style.**

To apply a color gradient to the background color, check the **Gradient Fill** check box and then click the **...** button. Click on the color box in the **Configure Gradient** dialog box to open the Color Palette.

Click the **Font** button to select a font and point size for text. If you want to preserve the proportion of the text size relative to the display when resizing the display, check the **Scale Font When Stretching Text** check box

Display Scaling allows you to choose between a **Fixed-Scale** display in which the objects will not change in size when the main window is resized, and a **Scalable Display** in which objects will automatically resize to fit within the main

window. The **Preserve Aspect Ratio** check box provides you with the option to scale the display in proportion to the screen's aspect ratio.

The **Display Dimensions** settings determine the **Width** and **Height** of the working area of the display.

If the **Query OPC data type and ranges during configuration** check box (**Application Preferences** dialog box only) is checked, then the system will look for OPC data types and ranges during the configuration of the application.

To make a backup copy of your files when saving your displays, check the **Make backup when saving displays** check box (**Application Preferences** dialog box only).

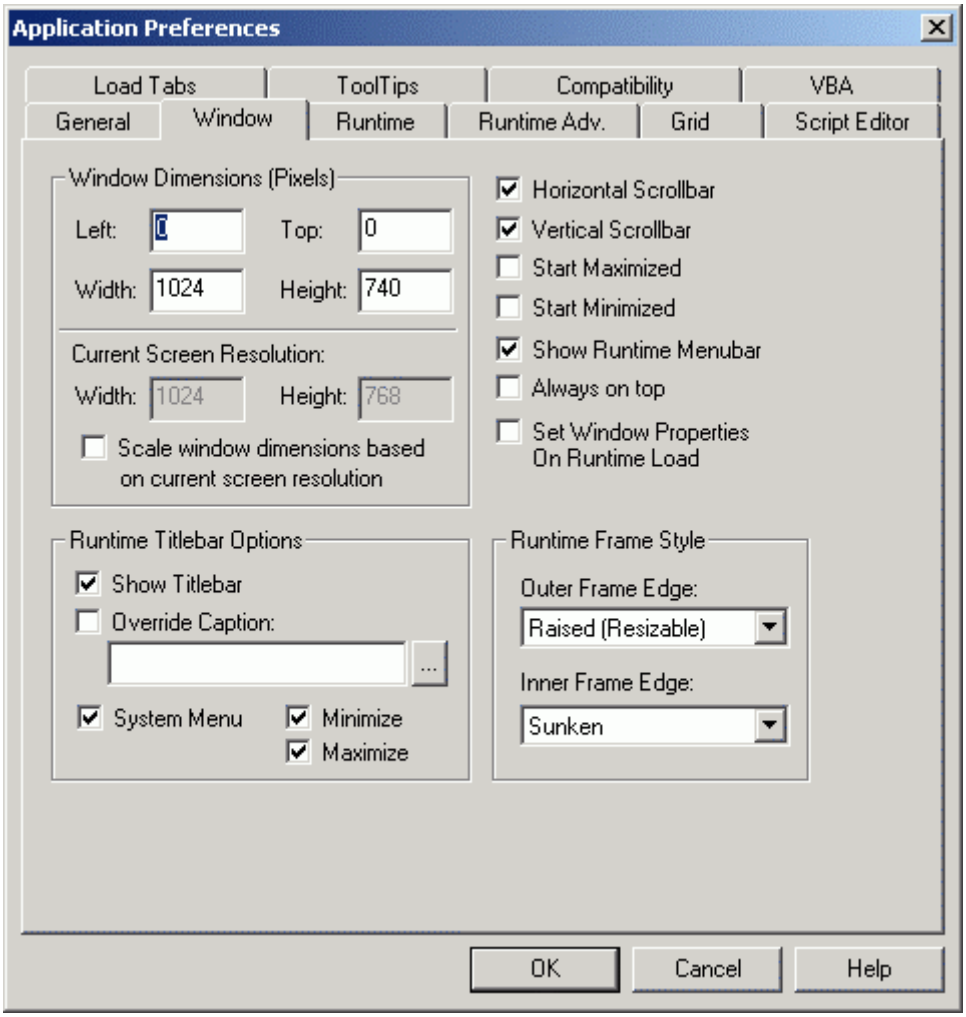
The **Save As Previous Version** feature allows you to save a display file created in a newer version of GraphWorX32 in one of the previous versions while preserving the existing version of the display file, so that the display file can be still loaded in the previous version of GraphWorX32 in which it was created. By default, GraphWorX32 provides warning messages about version compatibility. To suppress these version information messages when a display is loaded, check the **Load previous versions without warnings** check box (**Application Preferences** dialog box only). For more information, please see the Save As Previous Version section.

When publishing GraphWorX32 displays to a Web server, you must save the file as a "GraphWorX32 Display without VBA (*.gdf)," because the Web Publishing Wizard does not support Microsoft Visual Basic for Applications (VBA). In the **Application Preferences** and **Display Properties,** you can specify whether to save display files with or without VBA. Under the **Display Load/Save** section, select **VBA Display** if you want your displays to be saved automatically as a VBA display. Select **Non-VBA** if you want your displays to be saved automatically without VBA.

Window Options

The **Window** tab in the **Application Preferences** and **Display Properties** dialog boxes, shown below, allows you to configure parameters for the GraphWorX32 main window.

Note: Some of the window settings are only active in runtime mode.



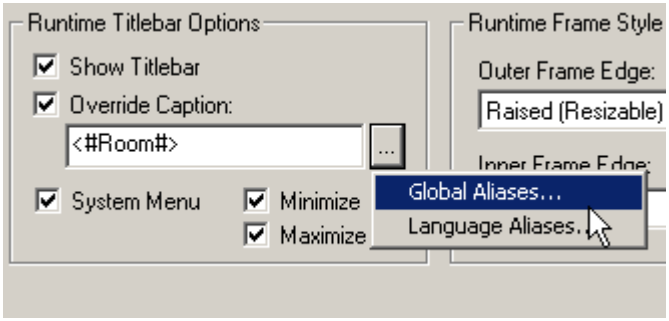
Application Preferences: Window Tab

The **Window Dimensions (Pixels)** field resizes the window to the exact dimensions desired (Left, Top, Width, and Height).

When the **Scale window dimensions based on current screen resolution** check box is checked, the window dimensions are automatically scaled based on the current screen resolution. For example, if the current screen resolution is 640x480 and your GraphWorX32 window size is 640x480, if you switch the window's resolution to 800x600 the next time your display is loaded, its dimensions will be scaled up to 800x600. If you want your window always to be 640x480 no matter what the resolution is, do not check this check box.

Note: If the screen resolution is changed, the window dimensions will remain the same as the way they were originally set up.

The **Runtime Title Bar Options** determine how the display window's title bar looks during runtime mode**.** You have the option to hide or show the title bar by unchecking or checking the **Show Title Bar** check box. To replace the title bar caption with a different caption, check the **Override Caption** check box and type a new caption in the text box below. To select an alias to use as a caption, click the **...** button to the right of the text box and select either **Global Aliases** or **Language Aliases** from the pop-up menu, as shown in the figure below. This opens the respective alias browsers, allowing you to choose from a list of available aliases. You can also **Minimize** or **Maximize** the **System Menu** by checking the check boxes provided.



Selecting an Alias To Use for the Runtime Title Bar Caption

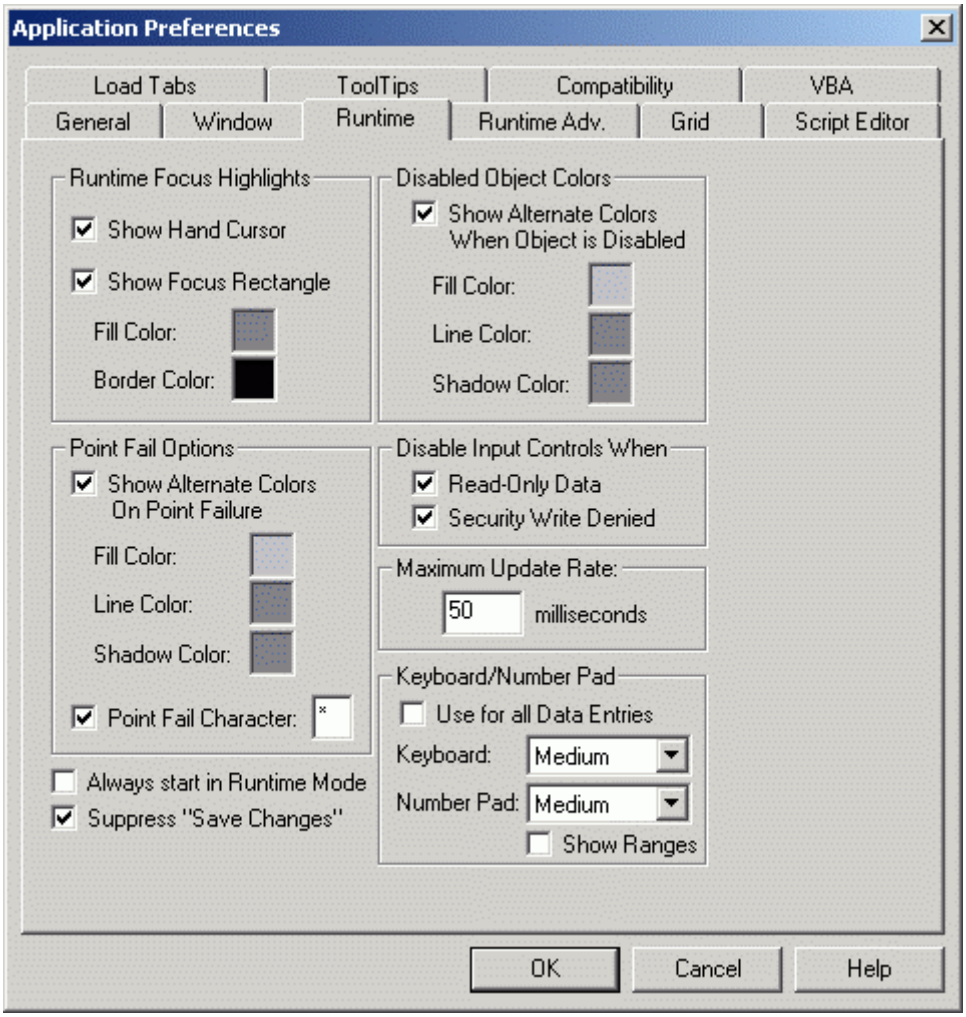
You may also choose to hide or show the **Horizontal/Vertical Scrollbars** (configuration mode and runtime mode), and the main **Menu Bar** (runtime mode only). You may choose to have the window start initially minimized or maximized. **Always on Top** ensures that the GraphWorX32 window will not be obscured by other open windows (runtime mode only).

Set Window Properties on Runtime Load indicates whether the window properties for a given display should be applied whenever that display is loaded while in runtime mode (when checked), or only when GraphWorX32 is initially launched into runtime mode (subsequent display loads will retain the window properties of the previously loaded display).

The **Runtime Frame Style** options allow you to specify styles for the inner and outer edges of the main window's border. In the list box under **Outer Frame Edge,** select None, Raised (Fixed Size), or Raised (Resizable). In the list box under **Inner Frame Edge,** select None or Sunken.

Runtime

The **Runtime** tab in the **Application Preferences** and **Display Properties** dialog boxes, shown below, specifies parameters for runtime mode.



Application Preferences: Runtime Tab

For **Runtime Focus Highlights,** you can choose to show a **Hand Cursor,** which indicates that you can pick the object. You can also choose to show a frame or a **Focus Rectangle** around the pickable object**,** and specify a fill color and a border color for the frame.

Point Fail Options allow you to choose whether to **Show Alternate Colors** when data points fail, by checking the box provided. By default, the **Point Fail Character** (shown for process points and data entries) is an asterisk as shown above.

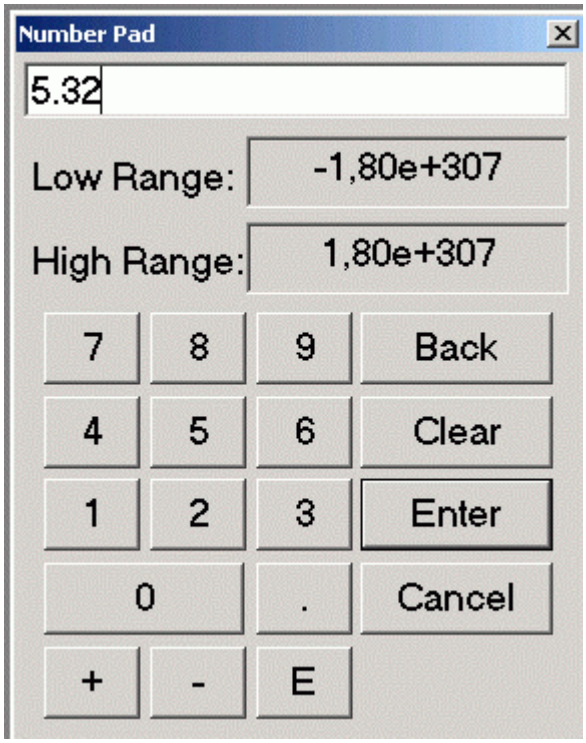
As with point fail options, the **Disabled Object Colors** specifies that you wish alternate colors to be shown when the object is disabled. Like disabled buttons in Windows, when an object is disabled in GraphWorX32, you cannot pick that object. **Data Input Controls** can be automatically disabled when the associated data connection is **Read-Only** or when **Security Write Access** is denied for the associated data connection.

The **Maximum Update Rate** reflects how fast runtime updates can occur. The fastest possible update rate is 50 milliseconds.

The **Always Start in Runtime Mode** check box (**Application Preferences** dialog box only) allows you to choose to start the application in runtime mode. Checking the **Suppress "Save Changes"** check box (**Application Preferences** dialog box only) saves the changes made in runtime without prompting the user to do so.

Keyboard and Number Pad

The **Keyboard/Number Pad** field (**Application Preferences** dialog box only) enables you to globally configure ranges for keyboards and number pads, such as the one shown below, for use with entering values to process points during runtime mode. If the entry is text, the keypad opens; if the entry is a number, the number pad opens. The available ranges in the **Keyboard** and **Number Pad** list boxes are Very Small, Small, Medium, Large, and Very Large. Checking/unchecking the **Show Ranges** check box shows/hides ranges in the number pad dialog box. The number pad can be optionally shown in the **Data Entry-Keypad** function of the **Process Point/Data Entry (PPT/DE)** tab of the **Property Inspector** dialog box.



Number Pad

Resizable Font

The fonts inside the keypad and the number pad are resizable. The font size is increased when size of the keypad increases and the font size is decreased when the size of the keypad decreases.

Caps Lock Switch and Support for International Languages

Some keys have been aggregated together on one unique key. When you push the **Caps Lock** key, the keypad layout changes and shows the new character in CAPS and the new special character (for example, the key <<a>> becomes <<A>> and the key <<2>> becomes <<@>>), as shown in the figure below. New keys have been added for German, Italian and French and other special characters.

The **International** key on the keypad acts like a check box (like the **Caps Lock** key); if you click it then it remains depressed until the next time you click it. When the **International** key is not pressed (default) the keypad hides all the buttons containing characters that do not appear on the default English keyboard layout. When the **International** key is pressed, then the keypad does not hide any keys and the full layout is displayed with, for example, the German, Italian, and French special characters, as shown in the figure below.



International Keypad

The language switching for the number pad has been fixed so that when you switch languages the separator is a decimal separator and the thousand separator changes. For example:

- US notation: 15,030.99
- European notation: 15.030,99

This is reflected in the number pad key, which is <<.>> (point) or <<,>> (comma) in function of the language currently selected in GraphWorX32.

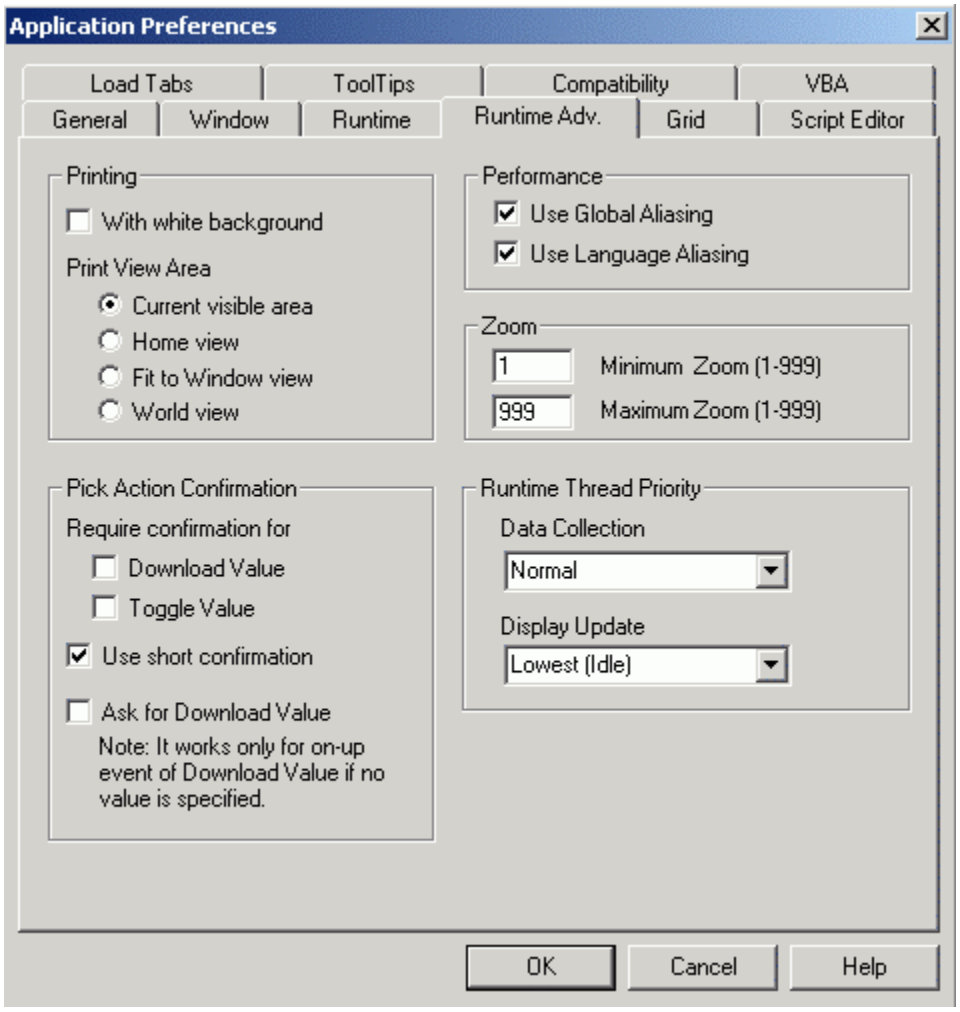
Customizing the Keypad

It is possible to customize the keypad through the Microsoft Visual Studio resource editor. You can specify up to two different characters for each key. Clicking the **Caps Lock** button switches between the two sets of characters. The first character is used when Caps Lock is clicked. The second character is used when Caps Lock is not clicked. The two characters do not need to be correlated: for example, <<2>> and <<@>>. If you do not specify a second character for a key, then the corresponding lower case system key will be used.

Runtime Advanced Settings

Application Preferences

The **Runtime Advanced** tab in the **Application Preferences** dialog box, shown below, specifies the following application parameters for runtime mode.



Application Preferences: Runtime Advanced Tab

Printing and Optional Print Area

The **Printing** field contains options for printing out a GraphWorX32 display. Checking the **With white background** check box overrides the default background color so that the display is printed with a white background.

The **Print View Area** options determine how much of the display will appear on the printout:

- Current visible area
- Home view
- Fit to window view
- World view

Pick Action Confirmation Messages

The **Pick Action Confirmation** field provides an option to require confirmation messages for the **Download Value** and **Toggle Value** pick actions. You can minimize the message length by checking the **Use Short Confirmation** check box.

Note: If no value is specified for the On-Up event, the Download Value pick action will display a number pad or a keypad dialog in runtime, where you can enter the value. This feature must be manually enabled in the **Runtime Advanced** tab of the **Application Preferences** dialog box by checking the **Ask for Download Value** check box.

Performance

The **Use Global Aliasing** and **Use Language Aliasing** check boxes under the **Performance** section are used to enable or disable support for global aliasing and language aliasing.

Zoom Preferences

The **Zoom** section allows you to specify the **Minimum Zoom** and the **Maximum Zoom** levels. It is possible to input any number between 1 and 999 for both the minimum and maximum zooms. If you enter a minimum zoom larger than the maximum zoom, GraphWorX32 automatically swaps the two numbers so that the maximum zoom is always greater than the minimum zoom.

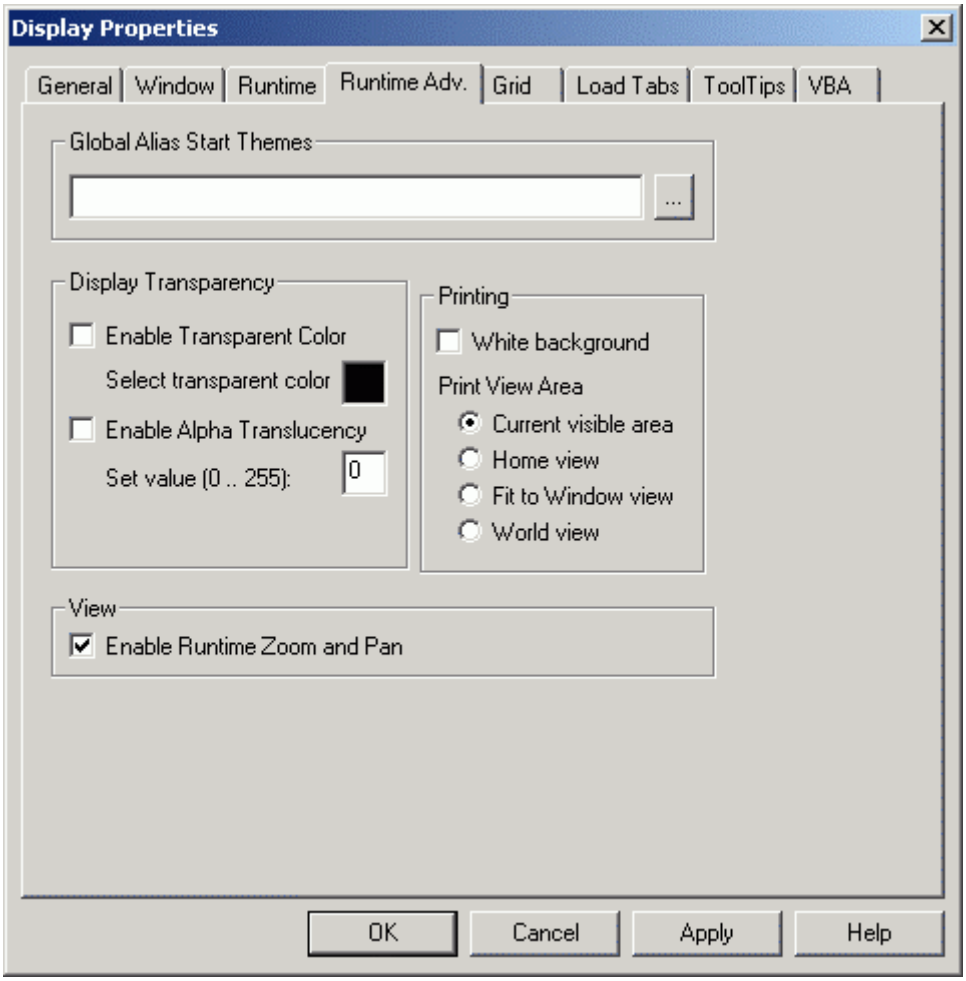
Runtime Thread Priority

The **Runtime Thread Priority** sets the priority levels for data collection and display updates during runtime mode. Select one of the following priority levels from each drop-down list under **Data Collection** and **Display Update:**

- Lowest (Idle)
- Low
- Below Normal
- Normal
- Above Normal
- High
- Highest (Time Critical)

Display Properties

The **Runtime Advanced** tab in the **Display Properties** dialog box, shown below, specifies the following display parameters for runtime mode.



Display Properties: Runtime Advanced Tab

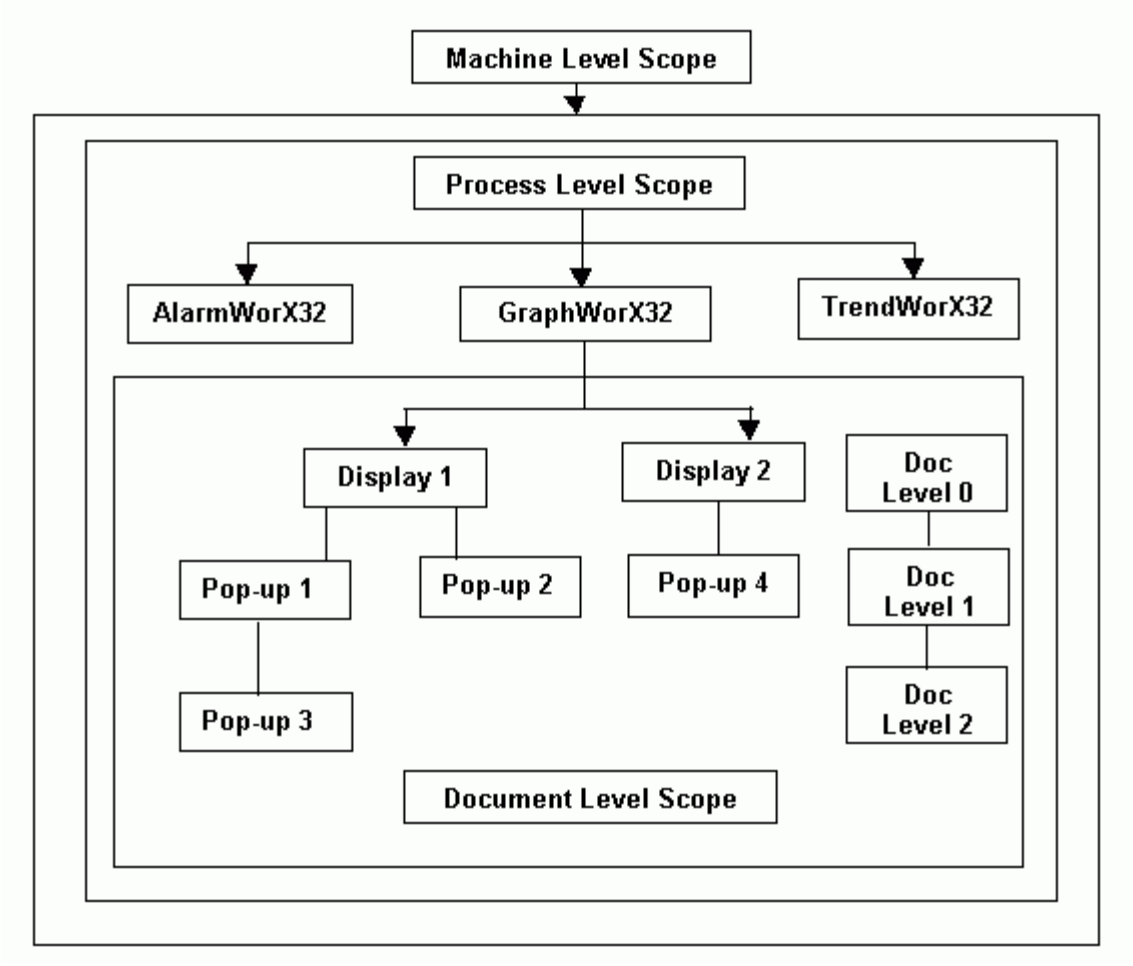
Global Aliasing Theme Scope

The resolution of aliases is controlled by the theme scope (i.e. on what level a theme is applied to your application). The Global Aliasing system supports three different levels of aliases, as shown in the figure below:

- **Machine-Level:** Applies to the whole computer.
- **Process-Level:** Applies to the current process (e.g. GraphWorX32 with pop-up windows, a TrendWorX32 pen, or an AlarmWorX32 Viewer).
- **Document-Level:** Applies only to the current document/display and its embedded controls (e.g. a GraphWorX32 pop-up window).
  Note: Document level aliasing is available only in GraphWorX32. It does not apply to the TrendWorX32 and AlarmWorX32 Viewer ActiveX controls.

Global aliases use the following syntax: **<#global_alias_name#>.**

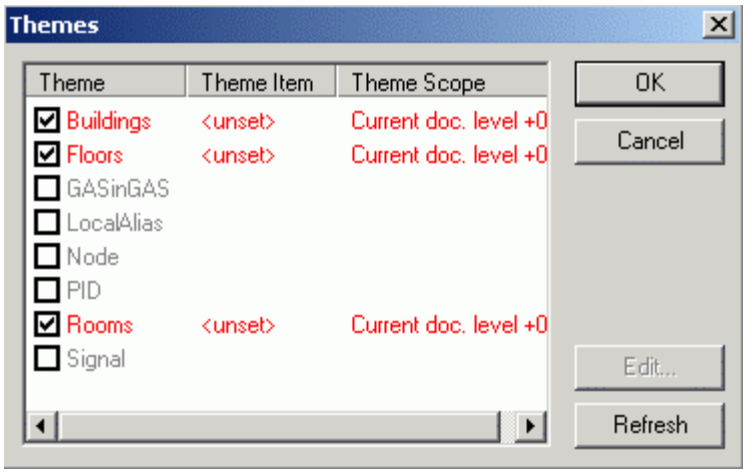
In GraphWorX32, you can have multiple levels of documents, or pop-up windows, as shown in the figure below.



Overview of Theme Scopes

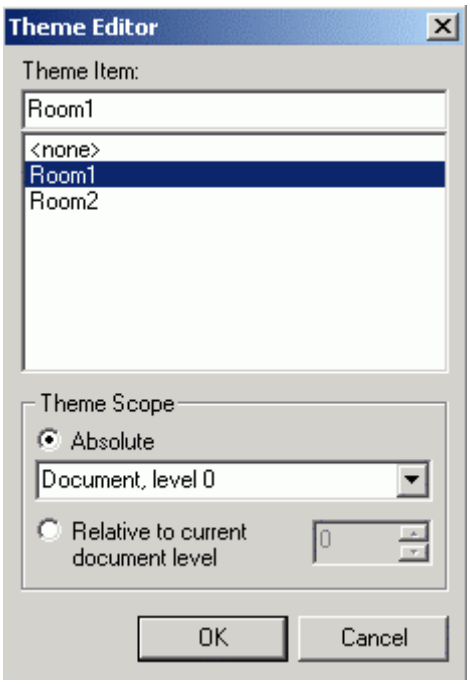
The document level theme scope is defined in the GraphWorX32 **Display Properties.** To define the default theme scope in GraphWorX32:

1. Either type a theme name in the text field under **Global Alias Start Themes** or click the **...** button to select a theme from the **Themes** dialog box, as shown in the figure below.



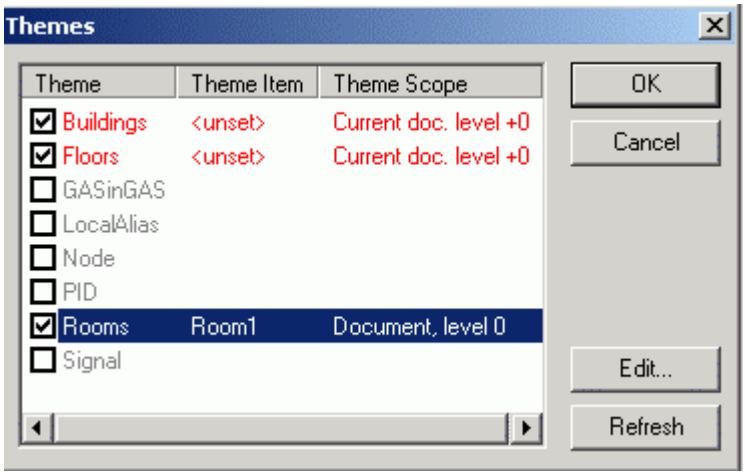
Choosing a Global Alias Start Theme

2. To define the theme scope, highlight a theme and click the **Edit** button. This opens the **Theme Editor** dialog box, as shown in the figure below. Select a **Theme Item** (e.g. "Room1") to associate with the start theme. Under **Theme Scope,** you can specify an **Absolute** theme scope (e.g. machine level, process level, or document level). Alternatively, you can specify a scope for the theme that is **relative to the current document level** (e.g. main display, pop-up window 1, pop-up window 2, etc.). Click **OK.**



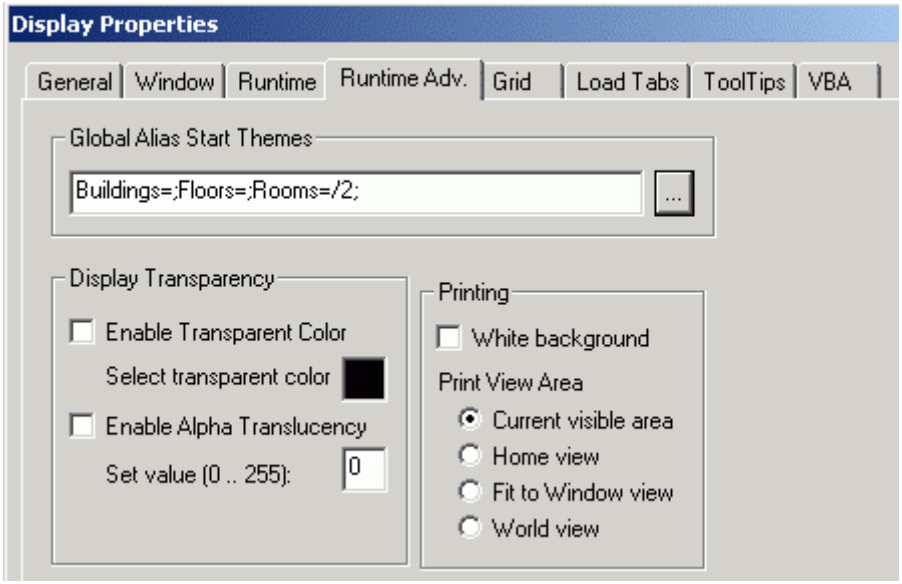
Specifying the Theme Item and Scope

3. The selected **Theme Item** and **Theme Scope** are now indicated in the **Themes** dialog, as shown in the figure below. Click **OK.**



Theme Scope Indicated

4. The selected themes appear under **Global Alias Start Themes** in the **Display Properties** dialog box, as shown in the figure below.



Global Alias Themes Added to Display Properties

Transparency and Translucency

GraphWorX32 (supported on Windows 2000 and later) enables defined runtime display transparency and translucency.

To enable display transparency, check the **Enable Transparent Color** check box. Choose the color that will be transparent during runtime mode (black by default) using the **Select transparent color** box.

To enable display translucency (also called alpha blending), check the **Enable Alpha Translucency** check box. In the **Set value** field, the translucency can be set in a range 0-255. When the alpha value is 0, the window is completely transparent. When the alpha is 255, the window is opaque.

In addition to setting the values in the **Display Properties,** the following automation methods, members of **Gwx32.GwxDisplay,** set the transparency in runtime:

- Function **SetTransparency**(EnableTransparency As Boolean, transparentColor As OLE_COLOR, EnableAlpha As Boolean, AlphaBlending As Integer) As Boolean
- Function **GetTransparency**(TransparencyEnabled As Boolean, transparentColor As <Unsupported variant type>, AlphaEnabled As Boolean, AlphaBlending As Integer) As Boolean

Printing and Optional Print Area

The **Printing** field contains options for printing out a GraphWorX32 display. Checking the **With white background** check box overrides the default background color so that the display is printed with a white background.

The **Print View Area** options determine how much of the display will appear on the printout:

- Current visible area
- Home view
- Fit to window view
- World view

View

Setting the view dimensions can be used to zoom and pan the view of a display during runtime (check the **Enable Runtime Zoom and Pan** check box). A mouse zoom feature simplifies the display navigation:

- To zoom in, press the **ALT** key and roll the mouse wheel forward.
- To zoom out, press the **ALT** key and roll the mouse wheel backward.
- To pan the screen press the **ALT** key, push the left mouse button, and drag the mouse.

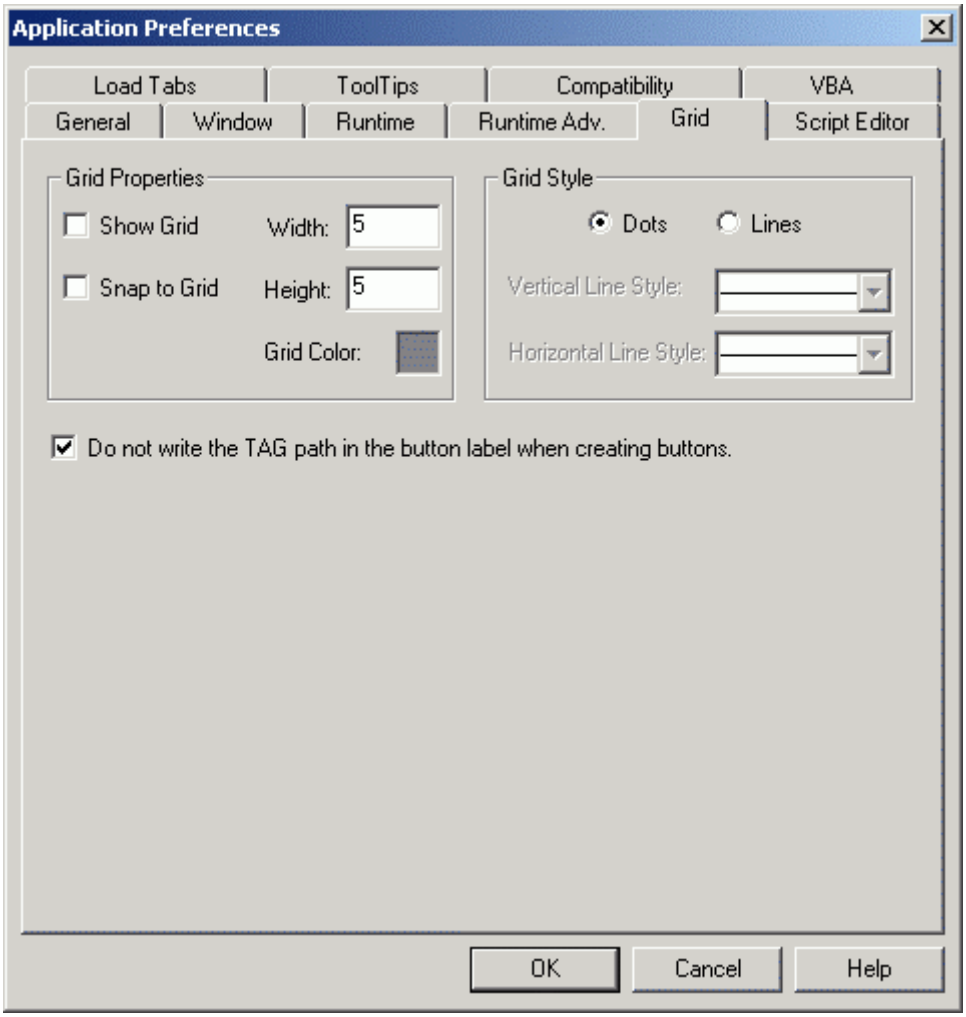
The zoom is selection-sensitive so that if an object is selected, the display automatically recenters the screen to the object. This also applies to multi-selection of display objects.

Grid

The **Grid** tab in the **Application Preferences** and **Display Properties** dialog boxes, shown below, gives you flexibility with line style, color, and spacing. The **Grid Size** is in arbitrary base units (depending upon the current zoom factor). When zoom is at 100%, the spacing is in pixels.

On the **Grid** tab, **Grid Style** allows you to choose a style of dots or dashes (lines). You can choose a **Vertical Line Style** or a **Horizontal Line Style** from the options available. The **Grid Properties** specify the grid spacing in terms of width and height.

You can choose a **Grid Color** for the lines and dots of the grid. By checking the appropriate boxes, you can choose to **Show Grid** and **Snap to Grid.** The **Grid** command on the **View** menu also shows or hides the grid.



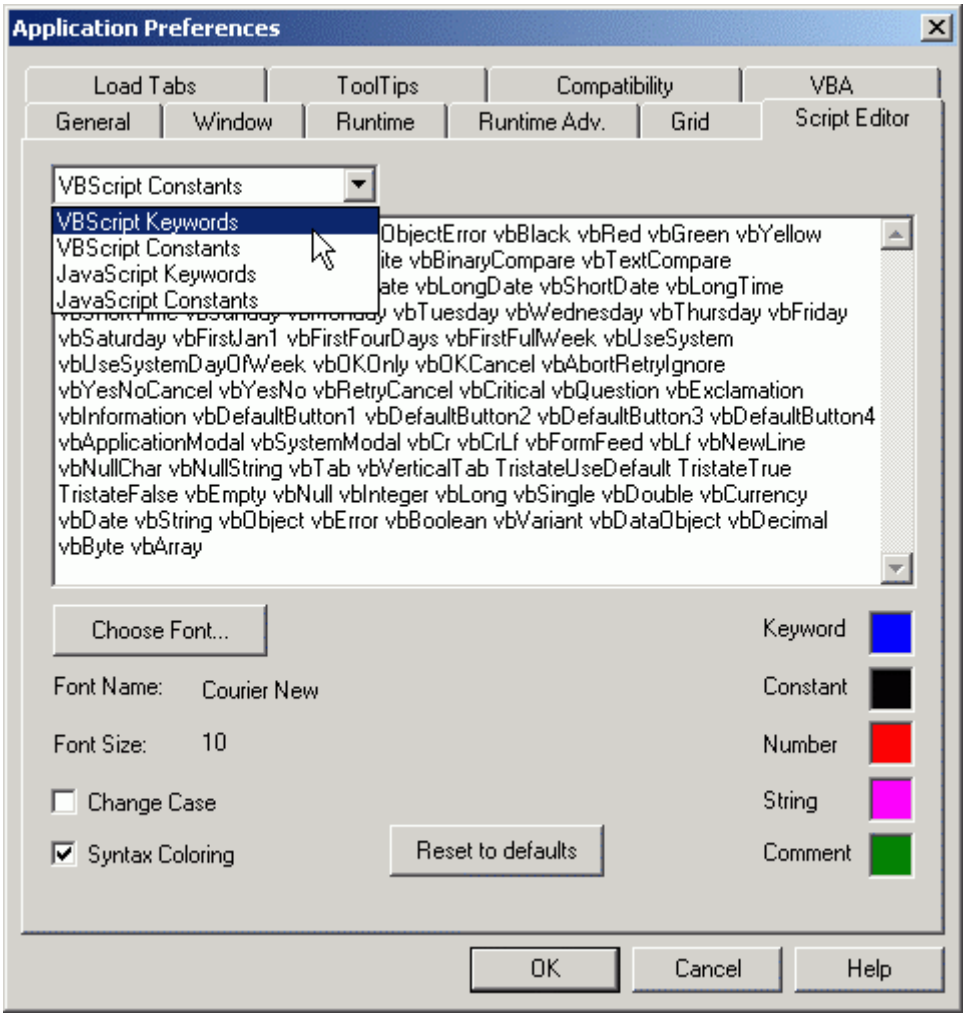
Application Preferences: Grid Tab

Script Editor

The **Script Editor** is designed to be customizable for users with special needs. The customization settings can be configured in the **Script Editor** tab of the **Application Preferences** dialog box. Select **Application Preferences** from the **Format** menu in GraphWorX32. This opens the **Application Preferences** dialog box, as shown in the figure below. Click on the **Script Editor** tab. You can change the following Script Editor settings, as shown in the figure below:

- Font face, size and style
- VBScript keywords and constants (select from the drop-down list)
- JScript keywords and constants (select from the drop-down list)
- Color for keyword, constants, string, number, and comment
- Enable or disable the syntax coloring
- Enable or disable the automatic case change feature

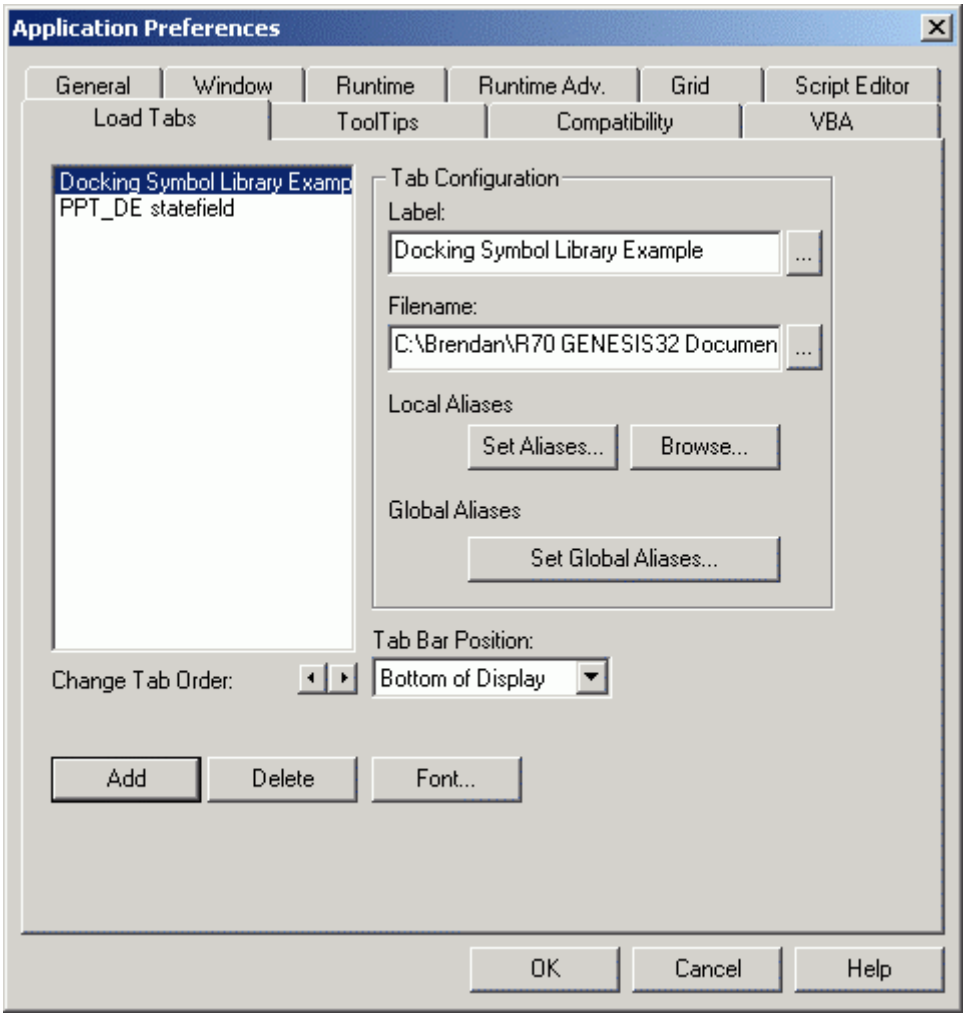
For more information, please see the **Scripting Engine and Script Editor for VBScript and JScript** section.



Application Preferences: Script Editor Tab

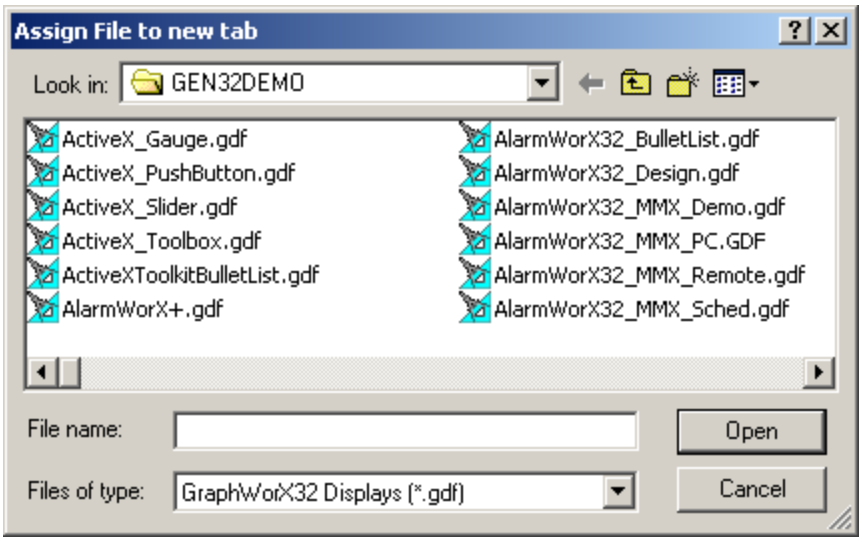
Load Tabs

The **Load Tabs** tab in the **Application Preferences** and **Display Properties** dialog boxes, shown below, allows you to construct a row of tabs; each tab will load a display file. Load display tabs are a convenient method for navigating through numerous display files.



Application Preferences: Load Tabs Tab

To add a tab, click **Add** and select one or more files from the **Assign File to New Tab** dialog box, as shown in the figure below. Click **Open.** Tabs are added in the list box on the left-hand side of the **Load Tabs** properties page, as shown above. To delete a tab from the list, click the **Delete** button. To change the position of the tabs in the list (move them up or down), click the arrow buttons next to **Change Tab Order.** You can also change an existing tab's settings by double-clicking on it inside the list box or by selecting the tab in the list and clicking the **Browse** button.

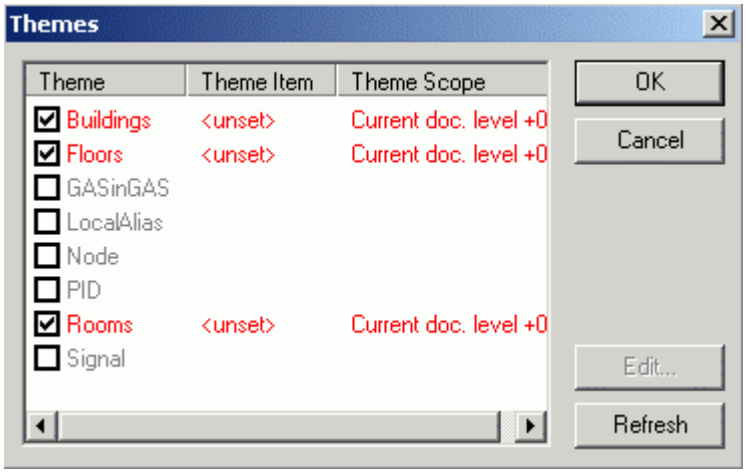


Assigning a File to a Tab

Select a **Tab Bar Position** at the top or at the bottom of the display. In the **Tab Configuration** section, type a **Label name** that for each tab that will identify the tab in the display. You can choose a **Font** from the options available by clicking on the **Font** button. The file name and directory path for each tag are listed under **File name.**

You can also specify both local and global aliases. Under **Local Aliases,** click the **Set Aliases** button. This opens the **Set Aliases Configuration** dialog box, where you can define the alias parameters. Under **Global Aliases,** click the **Set Global Aliases** button to select a theme from the **Themes** dialog box, as shown in the figure below. Click **OK.**

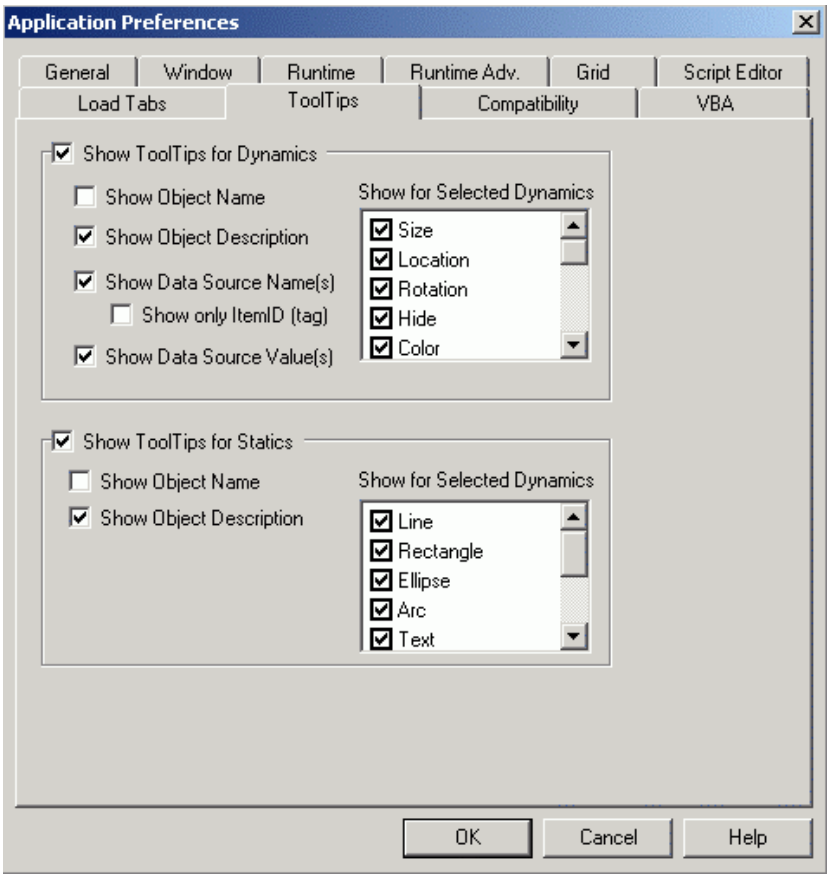
Note: To edit a theme's properties, select a theme and click the **Edit** button. This opens the **Theme Editor,** where you can specify theme items and the theme scope. For information about editing global alias themes and specifying the theme scope, please see the **Runtime Advanced Settings** section above.



Choosing Global Alias Themes

ToolTips

ToolTips are text messages that pop up when the mouse pointer is moved over certain objects in a display. The **ToolTips** tab in the **Application Preferences** and **Display Properties** dialog boxes is shown below.



Application Preferences: ToolTips Tab

ToolTips are enabled by default, but you can customize the ToolTip settings. The **Show ToolTips for Dynamics** section provides the following ToolTip options that apply to dynamic connections in the display (such as size dynamics or color dynamics):

- Show Object Name
- Show Object Description
- Show Data Source Name(s) (or show only the item/tag ID)
- Show Data Source Value(s)

The **Show ToolTips for Statics** section provides the following ToolTip options that apply to static objects in the display (such as rectangles and ellipses):

- Show Object Name
- Show Object Description

In the list boxes to the right, check each dynamic type (e.g. size, location, rotation, etc.) for which you want to display ToolTips.

Compatibility

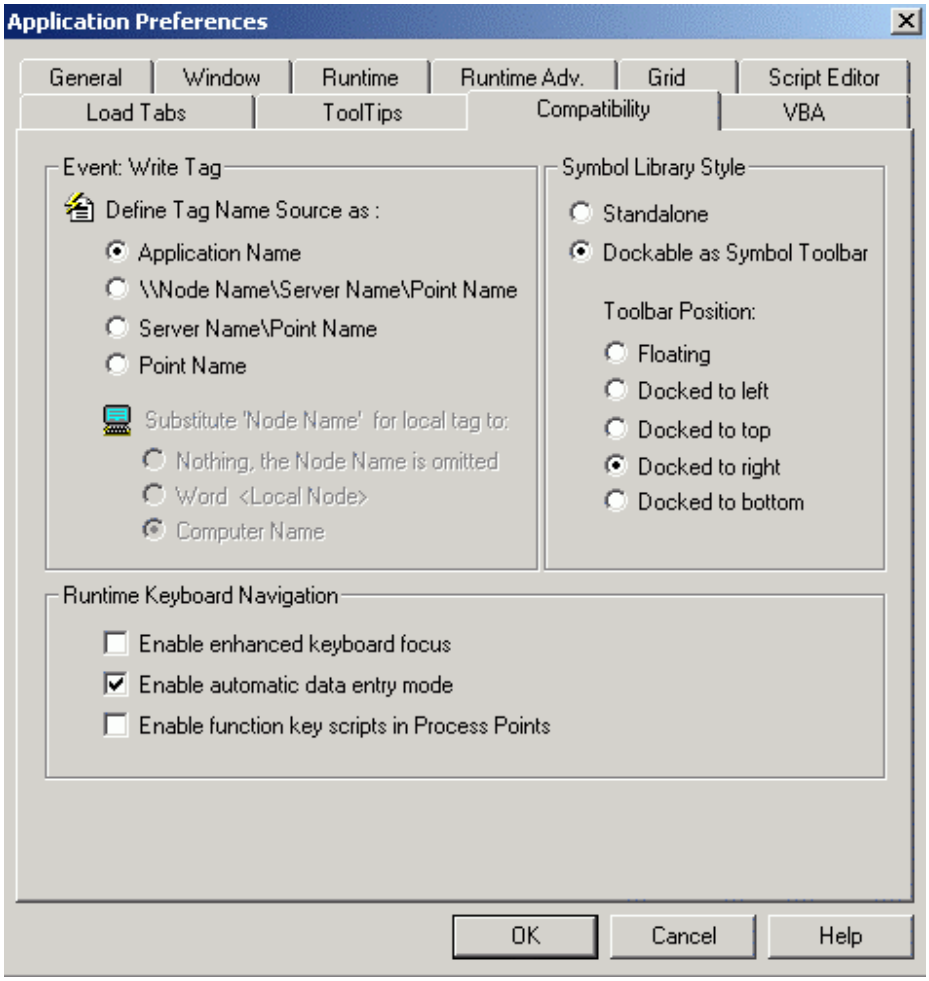
The **Event Log** enhancement adds more detailed event log reporting to GraphWorX32. In addition to existing log information, GraphWorX32 also logs the following actions:

- When GraphWorX32 starts
- When GraphWorX32 is shut down
- When a user saves a GraphWorX32 file

When a user saves a GraphWorX32 file, the message contains the full path of the file changed and the name of the user(s) that are logged into the Security Server at the time.

GraphWorX32 properly fills-in the **variant prevVal** in the Event Log structure when logging a writing to OPC data from a GraphWorX32 display, and it allows you to specify the "Source" name, which in previous versions of GraphWorX32 was always the name of the application. To configure the source type:

1. Select **Application Preferences** from the **Format** menu in GraphWorX32. This opens the **Application Preferences** dialog box, as shown in the figure below. Click on the **Compatibility** tab.



Application Preferences: Compatibility Tab

2. Select a source name, which can be one of the following values:
    - Application Name (just like it worked in earlier versions)
    - \\NodeName\ServerName\PointName (When this option is selected, you have the option to omit the "NodeName" from the string to substitute the "NodeName" with the local computer's name.)
    - ServerName\PointName
    - PointName
3. Click **OK.** The current configuration is stored to the registry.

Symbol Library Style

GraphWorX32 uses two different modes for the Symbol Library:

- Standard stand-alone Symbol Library
- Dockable symbol toolbar

The Compatibilty tab defines a default mode for the Symbol Library. If **Stand-alone** mode is active, clicking the **Symbols** button on the **Draw** toolbar launches the **SymbolLibrary.exe.** You can also select **Import > Symbol** from the **Draw** menu.

If the **Dockable as Symbol Toolbar** option is selected, then the Symbol Library is replaced with a Symbol toolbar inside the GraphWorX32 display, as shown in the figure below. You can also position the symbol toolbar as:

- A floating window
- Docked to left
- Docked to right
- Docked to bottom
- Docked to top

Runtime Keyboard Navigation

The **Runtime Keyboard Navigation** section contains the following check boxes:

- **Enable enhanced keyboard focus:** When checked, GraphWorX32 shows a focus more similar to Internet Explorer and other applications. The "keyboard" focus stays visible on the last selected input control, while hovering by mouse over other objects.
- **Enable automatic data entry mode:** When checked, the ProcessPoint control is automatically switched to input mode, so it is possible to write a value to it. Otherwise a control that gets a focus by keyboard must be activated by the **Enter** key for input.
- **Enable function key scripts in process points:** When checked, GraphWorX32 fires OnKey events that can be handled in VBA or VBScript/JScript.

In addition to the features described above, the following set of automation methods controls the runtime focus and simplifies the navigation from the script:

GwxDisplay

SetRuntimeFocus (ObjectName As String, bDataEntry As Boolean)
Sets the runtime focus to the visible object ObjectName which contains a dynamic action Pick, ProcessPoint, Slider, Dial.

SetRuntimeFocusNext (bWrap As Boolean, bDataEntry As Boolean)
Moves to the next tabable item, similar to pressing the Tab key.

SetRuntimeFocusPrev(bWrap As Boolean, bDataEntry As Boolean)
Moves to the previous tabable item, similar to pressing the Shift+Tab keys.

Object GetRuntimeFocus()
Returns a focused object.

CancelRuntimeFocus ()
Cancels the runtime focus.

GwxProcessPoint

SeRuntimetFocus (bDataEntry As Boolean)
Sets the focus on this particular PPT/DE object, only valid when this is a DE field.

GwxPick, GwxLocation, GwxRotation

SetRuntimeFocus ()
Sets the focus on this particular Pick, Slider or Dial object.

The parameters are defined as follows:

ObjectName
The object name of the visible object name which contains data entry dynamic actions.

bWrap
A boolean value, which allows the system to wrap around when tabbing, i.e. when at the first tabable item, goes to the last, and vice versa.

bDataEntry
A boolean value, which automatically opens the data entry for editing; this parameter is irrelevant when activating a pick field.

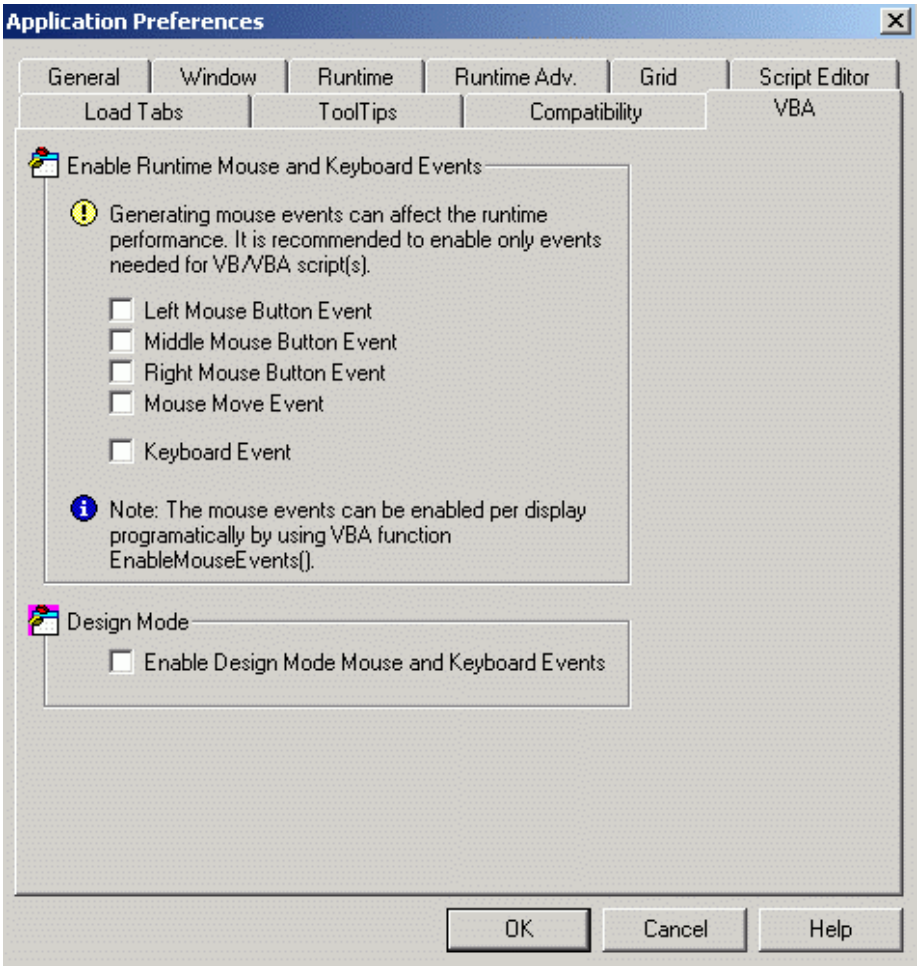
Visual Basic for Applications

Mouse and Keyboard Events

Early versions of GraphWorX32 (prior to version 5.0) contained mouse and keyboard events, which were fired to Visual Basic for Applications (VBA) and to the host container. Due to performance reasons, these events were removed in subsequent versions.

GraphWorX32 Version 7 enabled them again, but to avoid performance issues, events can be enabled individually, either globally or per display. The mouse and keyboard events can be selected globally for all instances of GraphWorX32 on given computer:

1. Select **Application Preferences** from the **Format** menu in GraphWorX32. This opens the **Application Preferences** dialog box, as shown in the figure below. Click on the **VBA** tab.
2. Select the desired mouse/keyboard events, which are then fired by GWX/GWXview32 to VBA or to the host container. Click **OK.** The settings are saved to the registry when GraphWorX32 stops and are restored when GraphWorX32 starts.



Application Preferences: VBA Tab

Configuring Runtime Events Per Display

The mouse and keyboard runtime events can be also selected locally for a particular display(s) by programmatically using the following automation function:

Sub EnableRuntimeEvents(LeftButton As Boolean, MiddleButton As Boolean, RightButton As Boolean, MouseMove as Boolean, Keyboard as Boolean)

A typical use of this method:

```
Private Sub GwxDisplay_PreRuntimeStart()
    ' Enable right click events only
    Call ThisDisplay.EnableRuntimeEvents(False, False,
True, False, False)
End Sub
```

Configuring Design Mode Mouse Events Per Display

Design time mouse events are enabled/disabled as a group per display. The default value can be set up in the **VBA** tab of the **Application Preferences** dialog box; then each new display inherits this value. The value can be changed in existing displays in the **VBA** tab of the **Display Properties** dialog box. There are also automation methods for modifying these events:

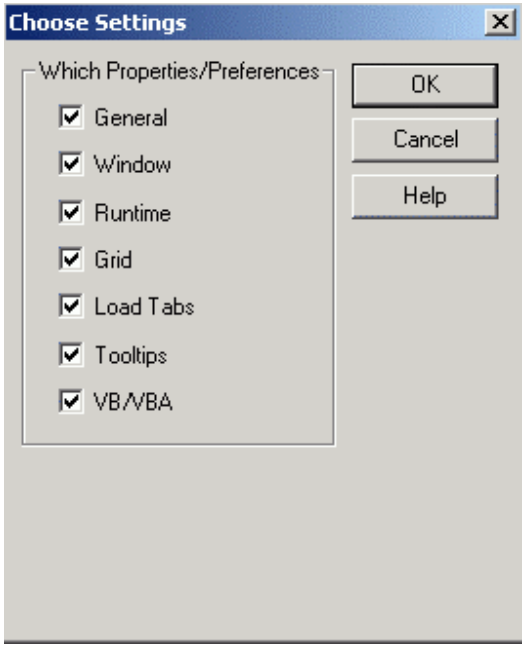
Sub SetDesignModeEvents(Enable As Integer)

Function GetDesignModeEvents() As Integer

Copying and Resetting Preferences and Properties

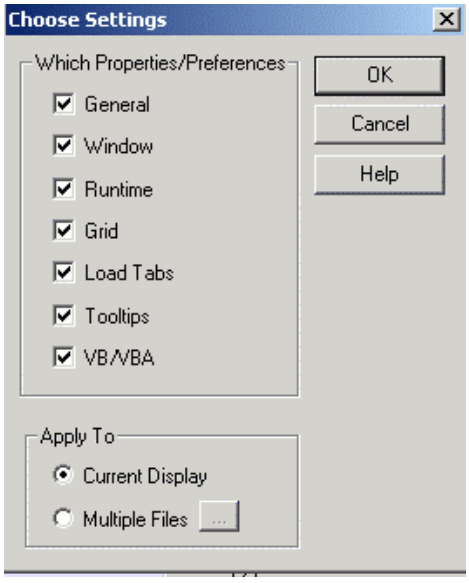
You can reset application preferences and display properties to their default settings by choosing the corresponding **Reset Default Application Preferences** or **Reset Default Display Properties** from the **Format** menu.

You can also copy settings from preferences to properties, or from properties to preferences. Choosing **Save Properties As Preferences** from the **Format** menu opens the **Choose Settings** dialog box, shown below.



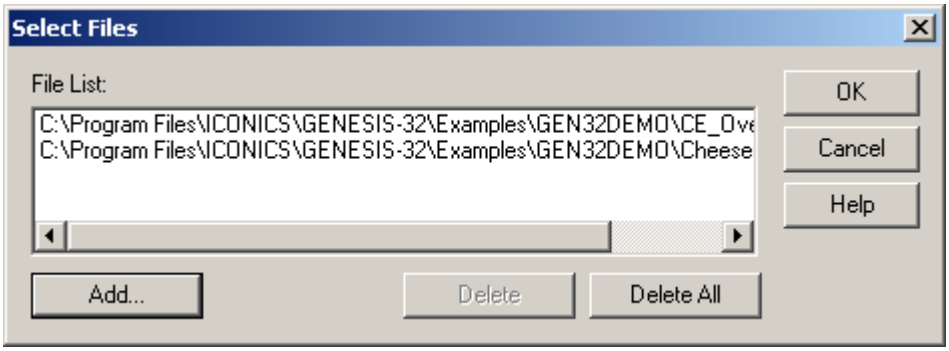
Choose Settings Dialog

Choosing **Apply Preferences to Properties** from the **Format** menu also opens the **Choose Settings** dialog box, as shown below.



Choose Settings Dialog

Each check box in the dialog corresponds to a properties/preferences category. Only those categories that are checked will be copied when **OK** is clicked. To apply the application preferences to the current display only, select **Current Display.** To apply the application preferences to more than one display, select **Multiple Files** and then click the **...** button. The **Select Files** dialog box opens, as shown in the figure below. Click the **Add** button to select files to which to apply application preferences. The files appear under the **File List** field.



Selecting Files for Applying Preferences

Chapter 3: Managing Display Files

Contents

Introduction
File Menu
GraphWorX32 File Name Extensions
Working With GraphWorX32 Display Files
Exiting GraphWorX32
Templates
Layers

Introduction

This section explains how to manage your GraphWorX32 display files. The **File** menu contains the functions that allow you to create, open, save, and print display files.

File Menu

The **File** menu commands are listed in the table below.

File Menu Commands

Command	Shortcut Keys	Function
New	CTRL+N	Creates a new display file.
Open	CTRL+O	Opens an existing display file.
Save	CTRL+S	Save the current display file.
Save As		Saves the display as a specified file type or version type.
Print	CTRL+P	Prints out the current display.
Print Preview		Shows you how a display will look on the printout before printing the display.
Print Setup		Configures the printer settings.
Exit		Closes the application.

GraphWorX32 File Name Extensions

In GraphWorX32, a file can be saved as a:

- GraphWorX32 display file (*.gdf) (with or without Visual Basic for Applictions)
- GraphWorX32 template file (*.tdf)
- GraphWorX32 display file for Windows CE (*.gdc)
- GraphWorX32 template file for Windows CE (*.tdc)
- Symbol file, or *.sdf

Working With GraphWorX32 Display Files

File-New



Select **New** from the **File** menu to create a new GraphWorX32 display file in your application.

File-Open



Select **Open** from the **File** menu to open an existing GraphWorX32 display file.

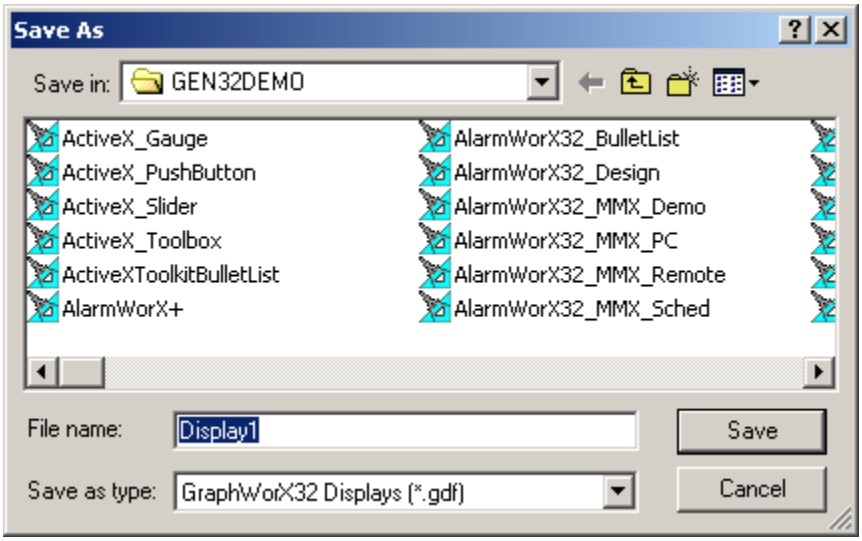
File-Save



Select **Save** from the **File** menu to save a GraphWorX32 display file.

File-Save As

To save the GraphWorX32 display as a specified file type or version type, select **Save As** from the **File** menu. This opens the **Save As** dialog box, as shown below.



Save As Dialog Box

The following options allow you to specify the name and location of the file you're about to save:

File Name

Type a new file name to save a document with a different name. A file name can contain up to eight characters and an extension of up to three characters. Your application adds the extension you specify in the **Save As Type** box.

Save As Type

The **Save As Type** box allows you to save as .gdf (normal) or .tdf (template). If you select an existing display name, it overwrites the display.

Note: Each display is saved as non-VBA until you execute a VBA-related command.

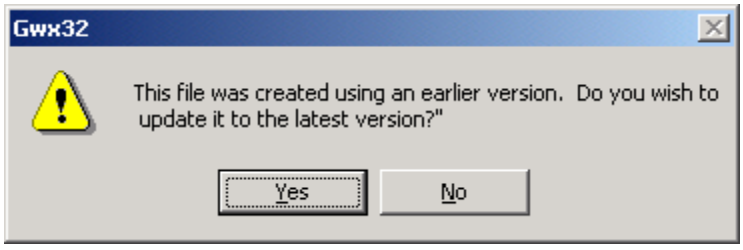
File-Save As Previous Version

The **Save As Previous Version** feature enhances the compatibility of current GraphWorX32 displays with previous versions of GraphWorX32. GraphWorX32 supports backward compatibility, which means that older displays can be loaded into a newer version of GraphWorX32. The Save As Previous Version feature allows you to save a display file created in a newer version of GraphWorX32 in one of the previous versions while preserving the existing version of the display file, so that the display file can be still loaded in the previous version of GraphWorX32 in which it was created.

Opening a Display File

When opening a display file, GraphWorX32 detects the current display version. If an earlier version of the display was detected, GraphWorX32 displays the following message, as shown in the figure below. You have two options: either update the display to the latest version of GraphWorX32 or keep the earlier version of the display.

- To update the display to the latest version, click the **Yes** button. All the new features of the latest version can be used and stored in the display. Such a display can be then loaded only in the latest version of GraphWorX32.
- To force GraphWorX32 to keep the earlier version of the display, click the **No** button. The next time you save the file it will be saved in that version, and then it can be loaded in a GraphWorX32 of that version or a newer version.



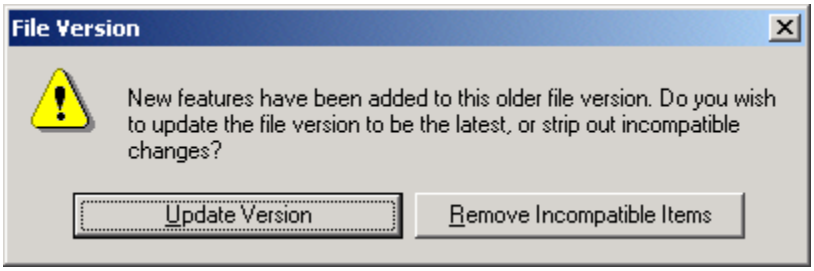
Message Indicating That the Display File Was Created Using an Earlier Version of GraphWorX32

Saving a Display File

When you save a display file, GraphWorX32 checks to see what features were added to the display and if those features are supported in the GraphWorX32 version that is currently selected. The current version is selected based on the loaded display file, and the version can be changed explicitly by using **File > Save As** dialog box.

If the version is the latest version, or if no unsupported features were added to the display file, GraphWorX32 saves the changes normally, without any warning message. If unsupported features were added to the display file, GraphWorX32 informs you about it and gives you two options, as shown in the figure below:

- To update the version to the latest file format and save all the new changes, click the **Update Version** button. (Such version cannot be loaded in the version of GraphWorX32 that was used to create the display.)
- To preserve the current display version and remove the items that made the display file incompatible with the current file version (i.e. features that did not exist in this version of GraphWorX32), click the **Remove Incompatible Items** button. By removing the incompatible items, the display is saved in the earlier version and therefore it can be loaded into that version and any later version of GraphWorX32.



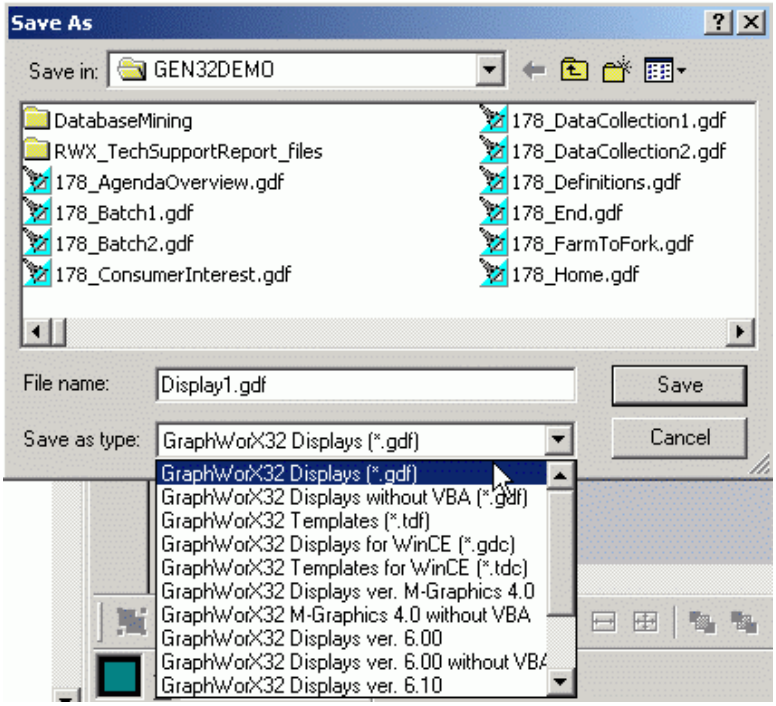
Option To Update Version of File or To Remove Incompatible Features

Saving a Display File As a Specific Version

To save a display file explicitly in one of the supported previous versions of GraphWorX32:

1. Select **Save As** from the **File** menu in GraphWorX32.
2. In the **Save As** dialog box, select a version of GraphWorX32 from the **Save as type** drop-down list, as shown in the figure below.
3. Click the **Save** button. The display is saved in the selected version of GraphWorX32. The version information for the file is preserved internally so that the next time you save the file by selecting **Save** from the **File** menu, the file will be resaved in the selected version.

   Note: The display can be saved either with VBA or without VBA. The non-VBA version is required by the GraphWorX32 Viewer ActiveX control and non-VBA GraphWorX32.

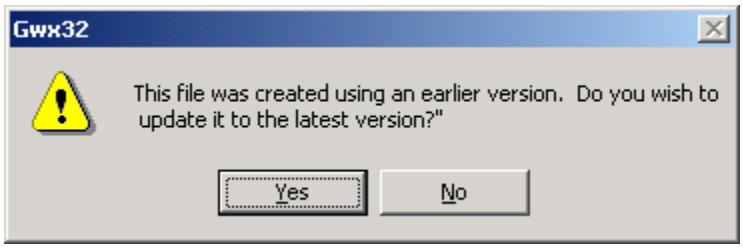


Saving a Display File as a Previous Version of GraphWorX32

Loading a Display in Runtime

When a display is loaded in runtime mode, the version information messages are suppressed but do not stop the runtime processing. However, when GraphWorX32 is switched to configuration mode, GraphWorX32 checks to see if the currently loaded display is the latest version or one of the earlier versions, and the message box appears again to let you decide which version to use. You have two options: either update the display to the latest version of GraphWorX32 or keep the earlier version of the display.

- To update the display to the latest version, click the **Yes** button. All the new features of the latest version can be used and stored in the display. Such a display can be then loaded only in the latest version of GraphWorX32.
- To force GraphWorX32 to keep the earlier version of the display, click the **No** button. The next time you save the file it will be saved in that version, and then it can be loaded in a GraphWorX32 of that version or a newer version.



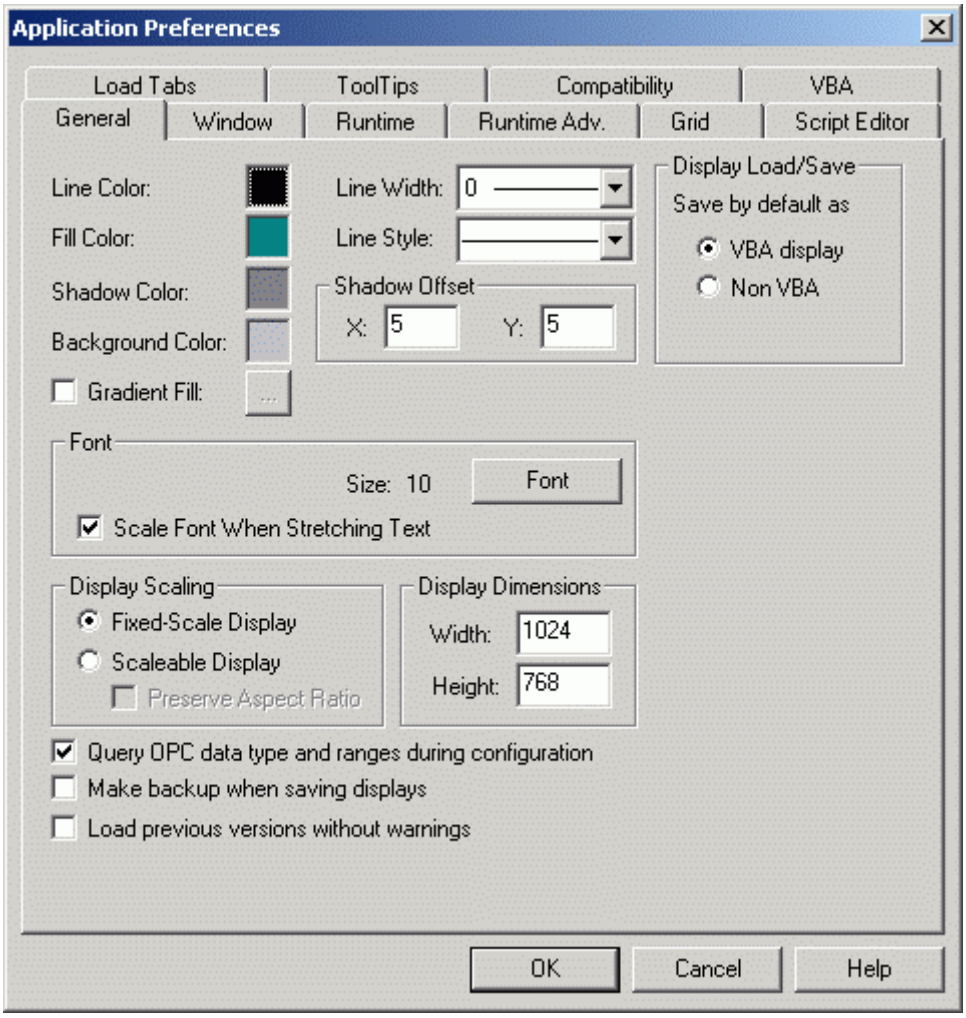
Message Indicating That the Display File Was Created Using an Earlier Version of GraphWorX32

Suppressing Version Compatibility Information

GraphWorX32 allows you to suppress the version information messages, working always with the latest version only.

To enable version suppression:

1. Select **Application Preferences** from the **Format** menu in GraphWorX32. This opens the **Application Preferences** dialog box, as shown in the figure below.
2. Click on the **General** tab. Check the **Load previous versions without warnings** check box. Click **OK.**
3. The value of the check box is automatically stored in the registry and is retrieved when either GraphWorX32 or this dialog box is loaded again.



Application Preferences: General Tab

Automation Enhancements

When the display is loaded or saved using automation methods, the warning messages are automatically suppressed so they do not interrupt the script. The following automation methods allow full control over the file versioning in automation:

- Function GetFileVersion() As long
- Function GetFileContainsVBA() As Boolean
- Function SetFileVersion(NewFileVersion As long) As long

The function **GetFileVersion()** returns the following values:

- -1 .. OpenFileError (&HFFFFFFFF)
- 0 .. Ver520
- 1 .. Ver600
- 2 .. Ver600SP
- 3 .. VerMGraphics40
- 4 .. Ver610
- 5 .. Ver620
- 9999 .. VerLatest (&H270F)

The function **GetFileContainsVBA()** returns True 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.

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).

Visual Basic 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
```

Known Limitations

Version Detection

The limitation is detecting the version of a display that contains only static objects and ActiveX controls but no dynamic objects. GraphWorX32 detects the version of the display based on the version of the most advanced dynamic object used in the display. If there are only static objects or ActiveX controls in the display, GraphWorX32 is not able to detect which version of the display this is and considers it to be the latest version.

Static Objects and ActiveX Controls

Static objects did not change between the GraphWorX32 6.0 Service Pack and the 6.10 versions, so such a display can be still open in the previous version of GraphWorX32.

ActiveX controls, however, keep the version within itself and GraphWorX32 does not have any control over it. The consequence is that **a display with embedded TrendWorX32 and AlarmWorX32 ActiveX controls fails to load properly in version 6.00 SP if it was saved in the latest version of GraphWorX32.**

Explanation of the Problem

In the case of ActiveX controls, GraphWorX32 asks each ActiveX control to serialize its content into the current display, but GraphWorX32 has no control over what is being serialized by each control. If a new version of the control is installed, it will apparently save in the latest version.

File-Print



To print a GraphWorX32 display, select **Print** from the **File** menu, or click the **Print** button on the **Main** toolbar. This command opens the **Print** dialog box, where you may specify the range of pages to be printed, the number of copies, the destination printer, and other printer setup options.

- **Printer:** This is the active printer and printer connection.
- **Name:** Select a printer from the list of available printers.
- **Properties:** Choose the **Properties** option to change the page layout and paper size.
- **Print Range:** Specifies the number and range of pages to print out.
- **Copies:** Specifies the number of copies to print out.

Shortcuts

Toolbar: 

Keys: **CTRL+P**

File-Print Setup

Selecting **Print Setup** from the **File** menu opens the **Print Setup** dialog box, which allows you to select a printer and printer connection. You can also specify the **Paper Size, Source,** and **Orientation,** as well as select the destination printer and its connection.

File-Exit

To close GraphWorX32, select **Exit** from the **File** menu. You can also use the **Close** command on the application **Control** menu. You will be prompted to save documents with any unsaved changes.

Shortcuts

Mouse: Double-click the application's Control menu button.

Keys: **ALT+F4**

Note: If you have not saved the changes in configuration mode and you exit during runtime mode, your changes will not be saved.

Templates

Templates are standard formats with established parameters that can be applied to a display at any time in the configuration mode.

GraphWorX32 supports two file extensions: .gdf (graphic display file), default file extension and .tdf (template display file).

The following functions are provided under the **Template** submenu of the **Format** menu:

- Apply Template
- Remove Applied Template
- Edit Applied Template
- Exit Edit Applied Template
- Insert Template Objects
- Update Template Displays

Apply Template

A template file can be applied to a display at any time in configuration mode. Select **Template > Apply Template** from the **Format** menu. Select a template (.tdf) file and then click **Open.** The specified template is loaded into the current display. Any objects already in the display are preserved. Also, the current summary information for the display is preserved. If you apply a template to a display that already has a template applied, the old template will be replaced by the new one. Objects in the applied template cannot normally be edited (they are essentially frozen background elements), however these objects can be modified using the "Edit Applied Template" functionality described below.

Remove Applied Template

To remove all the objects in a previously applied template, select **Template > Remove Applied Template** from the **Format** menu.

Edit Applied Template

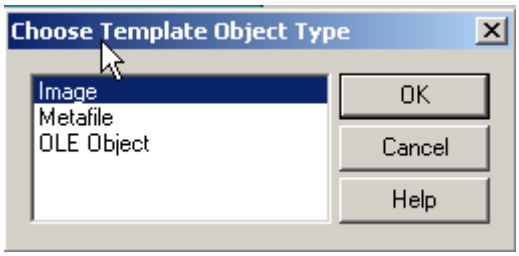
To edit the objects of an applied template, select **Template > Edit Applied Template** from the **Format** menu. When you enter this mode, only the objects in the template are visible (all other objects in the display are invisible and do not interfere with editing the template). Any edits made to the template's objects only affect the current display, not other displays that use the same template. To perform global edits on a template, see "Update Template Displays" below.

Exit Edit Applied Template

To cancel out of Edit Applied Template mode, select **Template > Exit Edit Applied Template** from the **Format** menu. You can also press the **Escape** key to exit Edit Applied Template mode.

Insert Template Objects

Selecting **Template > Insert Template Object** from the **Format** menu opens the **Choose Template Object Type** dialog box, shown below. Template objects are essentially placeholder frames that can later be replaced by a desired object. There are currently three types of template objects: image (e.g. bitmap, jpeg, etc.), metafile, and OLE object. When a template object is added to a display, it shows a text message like "Double Click Here to Add Image." (You can change this message.) When you double-click on the object, you are prompted for an image file that will replace the template object at its current size and location. This feature can be used to lay out generic displays that can be specialized later by filling in the placeholders for images, metafiles, and OLE objects.



Choosing the Template Object Type

Update Template Displays

To globally update displays that have the same template file applied as the one currently being edited, select **Template > Update Template Displays** from the **Format** menu. This function is only available when editing a .tdf file. You can edit a .tdf file and use this function to apply the changes to the displays that are based on that .tdf file.

Layers

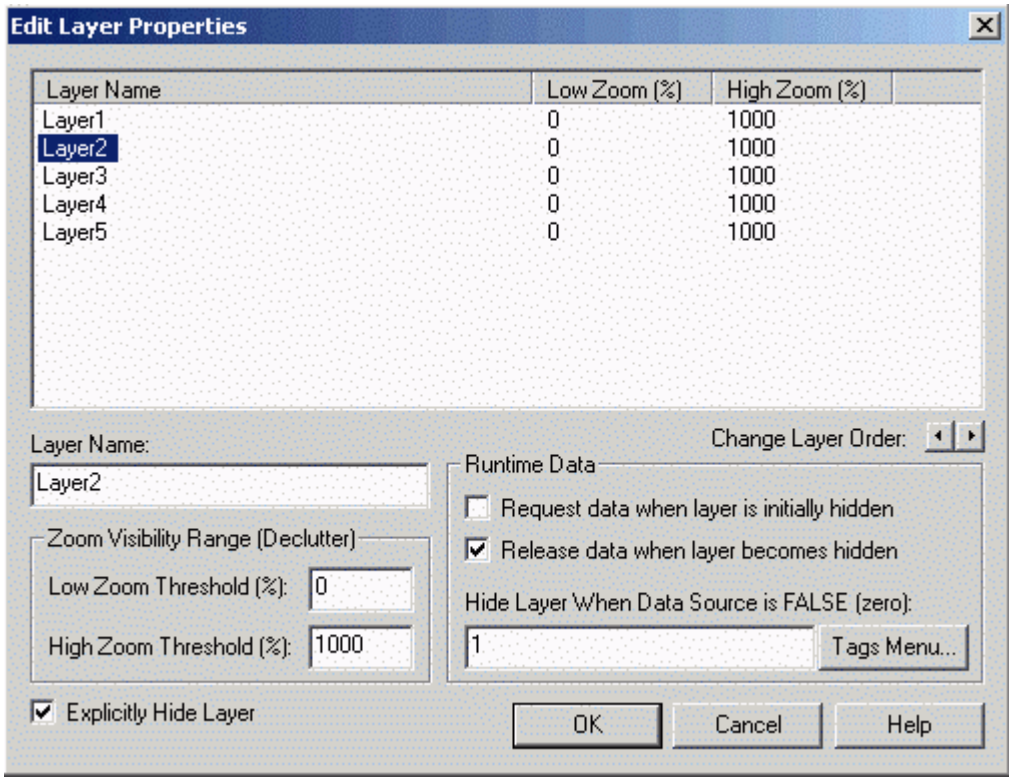
The use of layers in GraphWorX32 is helpful because it allows you to categorize elements of the display and separate it into levels of detail through decluttering. Using the **Layers** command on the **Format** menu, you can add, remove, or duplicate a layer. You can also edit the layer properties, set the active layer, set the next layer, set the previous layer, hide layers above the current layer and hide layers below the current layer.

Configuring Layers

Every GraphWorX32 display starts off as a display with one layer, the **primary** or system layer.

Note: This primary layer is the only layer to which a template can be applied.

It is important to understand the significance of the primary layer. When dealing with a display that has several layers, whatever you put in the primary layer appears in all layer views. While the objects placed in the system layer can be seen in the configuration of all other layers, it is not possible to edit these components while editing other layers. To add another layer, select **Layers > Add Layer** from the **Format** menu. This opens the **Edit Layer Properties** dialog box, shown below.



Edit Layer Properties Dialog Box

Through this dialog box, you can enter a layer name (this name should indicate what the layer contains or why it is a separate layer and should be unique to this display). Additionally, you can set a number of other configuration elements, including the zoom threshold and the runtime data acquisition elements.

Note: You can select an object in an inactive layer by pressing the **ALT** key and double-clicking the object.

Note: It is possible to change the properties once you have set them by selecting **Layers > Edit Layer Properties** from the **Format** menu.

Zoom Visibility Range (Declutter)

By assigning each layer a different zoom threshold, decluttering can occur. Decluttering involves automatically hiding or showing elements depending on the current zoom status. Zoom thresholds cannot be applied to the primary/system layer. The default values for zoom threshold are 0-1000, which makes the layer always visible. By changing this threshold for certain layers, the zoomed-out view will only show the most basic display.

Runtime Data

The **Runtime Data** section of the **Edit Layer Properties** dialog allows you to set how data should be retrieved from the display during runtime. If a layer has never been visible, you have two options: (1) to request data only when the layer becomes visible; or (2) to request data even while the layer is hidden so when the layer becomes visible it will contain valid data. If a layer was at one point visible and is no longer visible, you have the option of continuing to request data while the layer is hidden to maintain valid data or to release the data when the layer is hidden and request data when it becomes visible again.

Checking **Explicitly Hide Layer** hides the layer in runtime regardless of the **Hide Layer When Data Source Is FALSE (zero)** value. The **Explicitly Hide Layer** feature forces the layer to remain hidden regardless of whether the layer would be visible based on the zoom threshold. The layer will remain hidden until this option is turned off, at which point the layer may remain hidden depending on the attached tag value and zoom threshold. Some layers can hide appear based on other conditions, but an explicitly hidden layer is just hidden and never appears. If the layer is not explicitly hidden, then it may be hidden depending on the value of the attached tag and the zoom threshold (decluttering zoom).

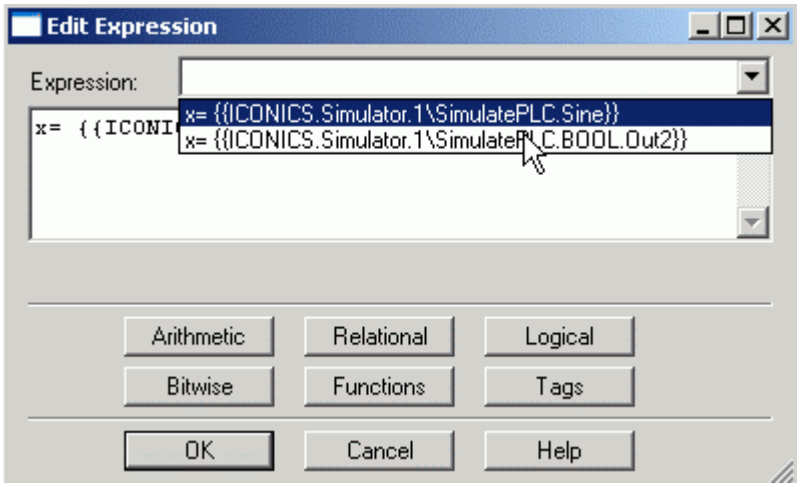
The **Tags Menu** button allows you to set each layer to hide when certain conditions become false (zero). This is helpful when dealing with a multilayer display that is only useful if you can see the active layer. The following options are available from the **Tags Menu** button.

Unified Browser. Selecting **Unified Browser** from the **Tags Menu** allows you to choose OPC tags, aliases, and variables from the **Unified Data Browser.** The browser shows all available data from the your machine as well as the network. Once you have selected the appropriate data type, click **OK** to return to the **Edit Layer Properties** dialog box.

Expression Editor. Selecting **Expression Editor** from the **Tags Menu** opens the **Edit Expression** dialog box. The **Expression Editor** is the same as the expression compiler used in many other areas of GraphWorX32. You can use

the Expression Editor to create equations, which may or may not contain OPC tags. Once the equation is set, click **OK** to return to the **Edit Layer Properties** dialog box.

Note: For more information about the Expression Editor, please see the Expression Editor help documentation.



Expression Editor

Layers Toolbar

The use of layers in GraphWorX32 is helpful because it allows you to categorize elements of the display and separate it into levels of detail through decluttering. Using the **Layers** toolbar, shown below, you can add, remove, or duplicate a layer. You can also edit the layer properties, set the active layer, set the next layer, set the previous layer, hide layers above the current layer and hide layers below the current layer. The **Layers** toolbar contains command buttons whose functions are identical to the commands under the **Format > Layers** menu (described above).

The **Layers** toolbar contains 10 numbered buttons (0-9) that allow you to switch between layers, as shown in the figure below. Clicking the **0** button always brings you back to the **Primary Layer.** Each time you create a new layer (in addition to the Primary Layer), a new number button is enabled for that layer. You can add a maximum of 9 layers (buttons **1-9**) that will be indicated in the **Layers** toolbar in addition to the Primary Layer (button 0). Any additional layers (i.e. more than 9), although present in the display, will not be indicated in the toolbar. Layers are added to the toolbar in the order in which they were created. Hovering the mouse over an active number button displays the layer number in a ToolTip.



Layers Toolbar

The **Layers** toolbar includes the following buttons:

- **Add Layer:** Creates a new user layer in the display in addition to the Primary Layer.
- **Remove Layer:** Opens the **Remove Layer** dialog box, allowing you to select which layer to delete from the display.

- **Edit Layer Properties:** Opens the **Edit Layer Properties** dialog box for the currently selected layer in the display.
- **Explicitly Hide Specific Layers:** Opens the **Check Layers to Hide** dialog box, allowing you to select which layer(s) to hide from the display. The **Explicitly Hide Layer** feature hides the layer in runtime and forces the layer to remain hidden regardless of whether the layer would be visible based on the zoom threshold. The layer will remain hidden until this option is turned off, at which point the layer may remain hidden depending on the attached tag value and zoom threshold. Some layers can hide appear based on other conditions, but an explicitly hidden layer is just hidden and never appears. If the layer is not explicitly hidden, then it may be hidden depending on the value of the attached tag and the zoom threshold (decluttering zoom). **Note:** You can also hide a layer by pressing the **CTRL** key and clicking on the layer's numbered toolbar button. For example, if you want to hide layer 7 only, press **CTRL** and simulatenously click on the number 7 button on the **Layers** toolbar.
- **Hide Layers Above Current Layer:** Automatically hides all layers above the currently active layer in the display (configuration mode only).
- **Hide Layers Below Current Layer:** Automatically hides all layers below the currently active layer in the display (configuration mode only).
- **Set Currently Active Layer:** Opens the **Set Currently Active Layer** dialog box, where you can specify the current editable layer. **Note:** The Currently Active Layer defaults to the layer that was active when the display was last saved.
- **Set Active Layer Previous:** Sets the editable layer to the layer that is just before the currently displayed layer. For example, if the current layer is Layer 3, the editable layer will be set to Layer 2.
- **Set Active Layer Next:** Sets the editable layer to the layer that is just after the currently displayed layer. For example, if the current layer is Layer 1, the editable layer will be set to Layer 2.

Editing Tools

There are two main editing tools that should be noted for the configuration of GraphWorX32 displays containing layers. The first is the **Hide Layers** feature in the **View** menu. The other is the **View** toolbar, shown below. The first option is also available in runtime mode and is discussed in greater detail in the following section.



View Toolbar

Using the **View** toolbar, you can change the view of the display by activating different zoom functions. Additionally, the toolbar provides specific shortcuts for layering. All of the following functions can also be found by selecting **Layers** from the **Format** menu.

Set Active Layer Previous

Selecting **Layers > Set Active Layer Previous** from the **Format** menu sets the editable layer to the layer that is just before the currently displayed layer. For example, if the current layer is Layer 3, the editable layer will be set to Layer 2.

Note: To change the order of the layers, select **Layers > Edit Layer Properties** from the **Format** menu and use the **Change Layer Order** arrow buttons located underneath the layer list box. The active layer name is always displayed in the right corner of the status bar.

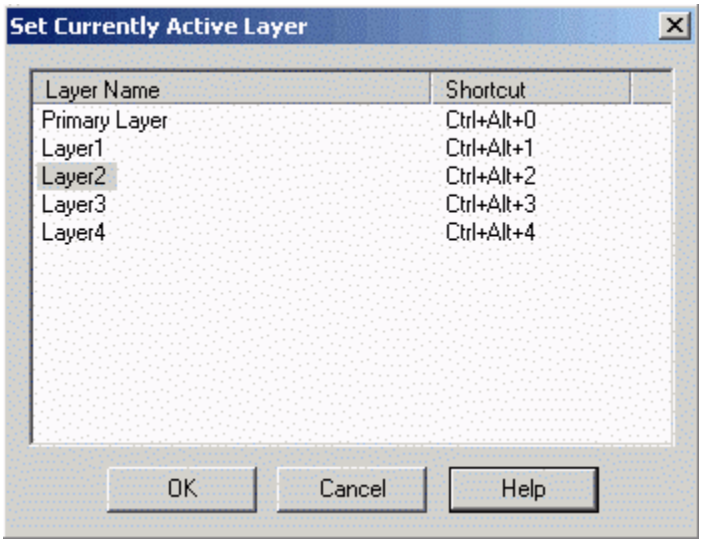
Set Active Layer Next

Selecting **Layers > Set Active Layer Next** from the **Format** menu sets the editable layer to the layer that is just after the currently displayed layer. For example, if the current layer is Layer 1, the editable layer will be set to Layer 2.

Set Currently Active Layer

Selecting **Layers > Set Currently Active Layer** from the **Format** menu opens a dialog box listing all possible layers, as shown in the figure below. Select the layer you want to edit, and then click **OK.**

Note: The Currently Active Layer defaults to the layer that was active when the display was last saved.



Setting the Currently Active Layer

Security for Layers

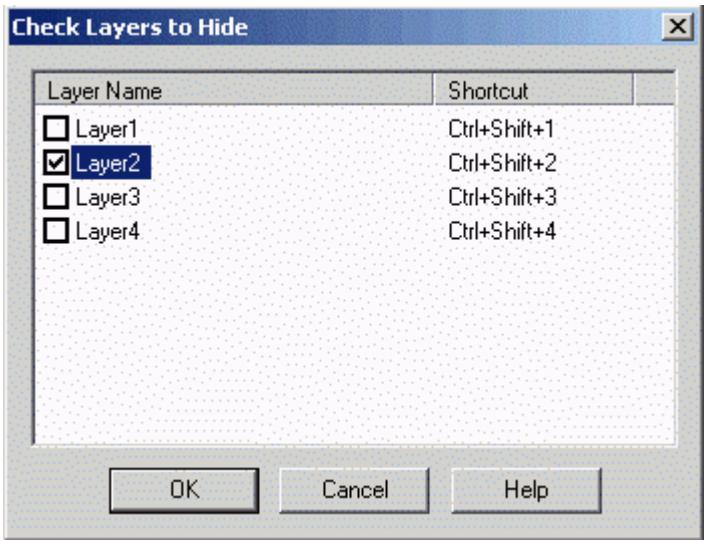
It is possible to set security for a specific layer within all displays, all layers within a specific display, or all layers in all displays. To set security for a layer, select **Security Configuration** from the **Tools** menu. This opens the Security Server login dialog box. The format for configuring security for layers is **filename|layername (ex. filename<vertical.pipe>layername)**.

It is also possible to use "wildcards" when formatting security for layers. The following table provides examples of how to use "wildcards."

Where To Set Security	Example
Specific layer in all displays	*.gdf|MyLayer1
All layers in a specific display	MyDisplay1|*
All layers in all displays	*.gdf|*

Using Layers in Runtime Mode

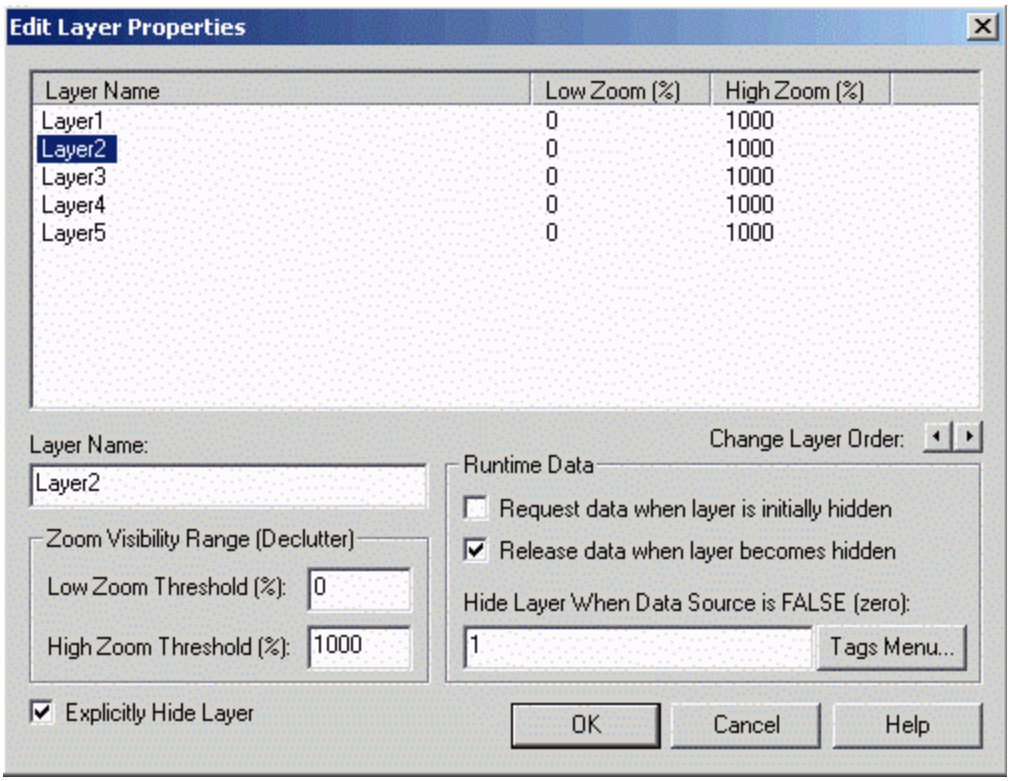
You can hide or show layers by selecting **Hide Layers** from the **View** menu or the keyboard shortcut combination **CTRL+SHIFT+0**. Both of these options open the **Check Layers to Hide** dialog box, shown below. Select a layer or layers to hide, and then click **OK.**



Selecting Which Layers to Hide

Layers may also be hidden due to the value of the tag to which they are attached, as well as if you select **Layers > Edit Layer Properties** from the **Format** menu and select the **Explicitly Hide Layer** check box in the **Edit Layer Properties** dialog box, as shown in the figure below. The **Explicitly Hide Layer** feature forces the layer to remain hidden regardless of whether the layer would be visible based on the zoom threshold. The layer will remain hidden until this option is turned off, at which point the layer may remain hidden depending on the attached tag value and zoom threshold. Some layers can hide appear based on other conditions, but an explicitly hidden layer is just hidden and never appears. If the layer is not explicitly hidden, then it may be hidden depending on the value of the attached tag and the zoom threshold (decluttering zoom).

Each layer that you added during configuration is shown in the dialog box, along with the system generated shortcut key. Note that there is a check box to the left of each layer name. If this box is checked, that layer will be hidden during runtime mode.



Explicitly Hide Layers Function in Layer Properties

Additional VBA Events for Layering

Event PreAnimateLayer(LayerName As String)	Fired before data for the layer are requested.
Event PostAnimateLayer(LayerName As String)	Fired after data for the layer have been requested.
Event PreDeanimateLayer(LayerName As String)	Fired before data for the layer are released.
Event PostDeanimateLayer(LayerName As String)	Fired after data for the layer have been requested.

Chapter 4: Creating and Modifying Objects

Contents

Introduction
Configuration Mode
Draw Functions
Property Inspector
Import Functions
Export Functions
Symbols
Formatting Objects
Arranging Objects
Rotation

Introduction

This section explains how to create and configure objects to use in your GraphWorX32 displays. You can draw objects, such as circles, boxes, and ellipses, using the drawing tools available from the **Draw** menu. You can also define various formats for these objects, including line style and weight, text fonts, custom colors and background colors, using the **Format** menu.

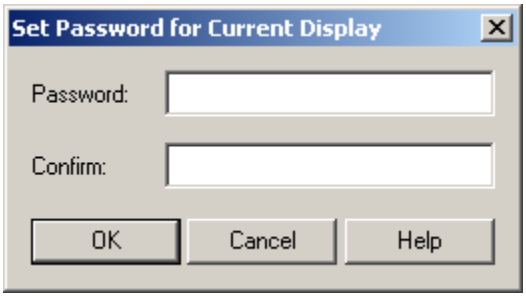
Configuration Mode

Configuration mode is the mode in which you design displays. In this mode, you can create static and dynamic objects, set general display properties, etc. Static objects are typically created on screen and then modified via the property inspector (some static objects, such as polylines, arcs, and text, also have additional on-screen editing capabilities). Dynamic objects are also configured via the Property Inspector.

Display Configuration Mode Password
Arrow Key Resize
TAB Key Object Selection
Alt Key Group Selection
Ctrl +Shift Drag
Template Activity
Find and Replace Features
Customize Toolbars

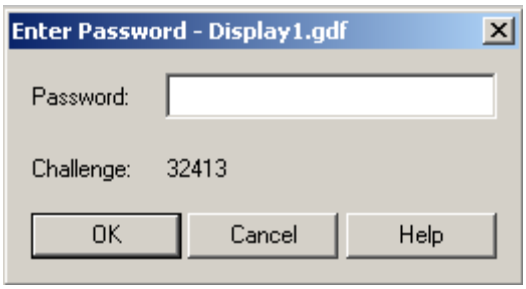
Display Configuration Mode Password

It is possible password-protect the configuration of a GraphWorX32 display. To do so, select **Set Configuration Mode Password** from the **Format** menu. This opens the **Set Password for Current Display** dialog box, as shown in the figure below.



Set Configuration Mode Password

Enter your password in the **Password** field, and then re-enter it in the **Confirm** field. When a user tries to open a password-protected display, that user will be presented with the following dialog before being able to enter the configuration.



Opening a Password-Protected Display

If the correct password is entered, the display opens in configuration mode. If an incorrect password is entered, a warning message is displayed, and the user will continue to be denied access until the proper password is entered.

Note: The display password does not apply to displays opened in runtime mode.

Arrow Key Resize

While in configuration mode, you can change the size and shape of an object using the arrow keys along with the **SHIFT** or **CTRL+SHIFT** keys. Holding down the shift key and pressing the arrow keys adjust the top and right sides of the object. Holding down the **CTRL+SHIFT** key combination and pressing the arrow keys adjust the bottom and left side of the object.

TAB Key Object Selection

You can use the **TAB** key to move the selector from one object to another. The selector moves around the display in the order in which the objects were created; or, if you have placed the objects in a particular order (i.e. brought an object forward or sent one back), then the selector moves from the object farthest (or the selected object) forward.

ALT Key Group Selection

If you press the **ALT** key and try to select a single object, you will notice that nothing happens. This is because the **ALT** key forces you to use the group select function. For the group select function, the select function is active and you draw a rectangle encasing at least one object, in essence selecting all objects entirely enclosed in the rectangle.

Note: An object is selected when the selector handles are visible around the edges of the object.

CTRL+SHIFT Drag

When you press **CTRL+SHIFT** and drag an object, the object will not only be copied, but you will only be able to drag the object in a horizontal or vertical path from the source location. This is helpful when you are trying to line objects up in the same axis.

Template Activity

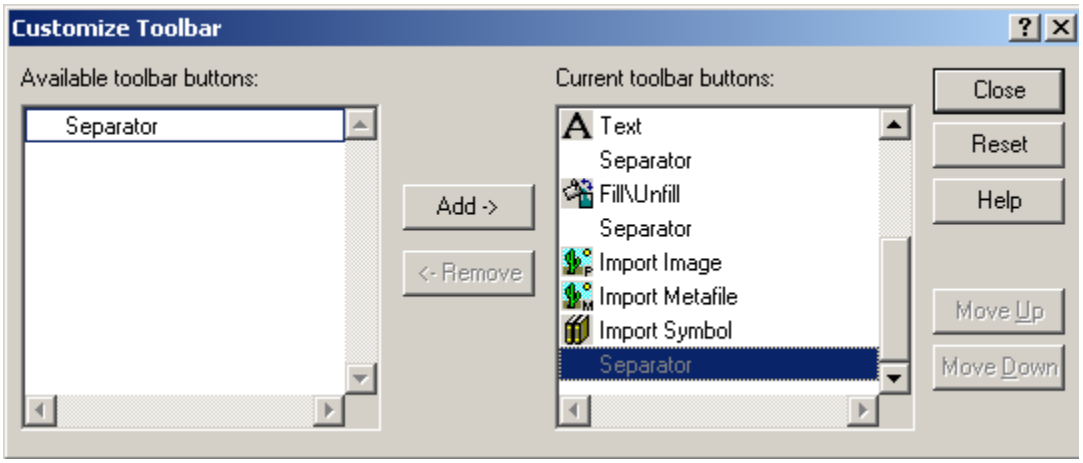
When a template is changed and saved, you are automatically asked if you wish to have all displays using the template to be updated as well. This is a very effective way to incorporate changes that may affect several displays. Although this may take some foresight on your part, effective planning can save you a lot of time in the long run.

Find and Replace Features

The GraphWorX32 **Find and Replace** function supports searches based on **Custom Data, Description** and **File Name** (for pick dynamics).

Customize Toolbars

You can add, remove, or relocate items on the various toolbars within GraphWorX32. Simply right-click on the toolbar you wish to edit. This opens the **Customize Toolbar** dialog box, as shown in the figure below.



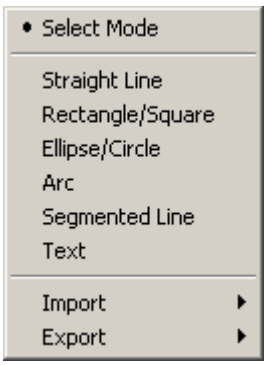
Customizing Toolbars

To remove a button from the toolbar, select the button from the list on the right under **Current toolbar buttons** and click the **<-Remove** button. Similarly, if you wish to add a button to the toolbar, select the button from the list on the left under **Available toolbar buttons** and click the **Add->** button. Use the **Move Up** and **Move Down** buttons to relocate buttons on the toolbar. This feature is helpful since there may be functions represented on a toolbar that you do not use (in which case it should be removed) or the default ordering of the buttons does not make any sense to you.

Note: All functions pertaining to a particular toolbar will be in either the list of available buttons or in the list of toolbar buttons. It is impossible to entirely remove a function from the dialog box.

Draw Functions

The **Draw** functions enable you to create display objects using various drawing tools. You can create complex drawings by grouping different display objects. The Draw functions are located in the **Draw** menu and the **Draw** toolbar, as shown in the figures below. The information on the bottom left of your screen corresponds to the Draw toolbar.



Draw Menu



Draw Toolbar

Selector Tool



With the **Selector** button, you can choose objects to modify. When you select an object, you can change its appearance or location. Before you can modify an object, you must select it. Once it is selected you can format and edit that object.

To select a single object:

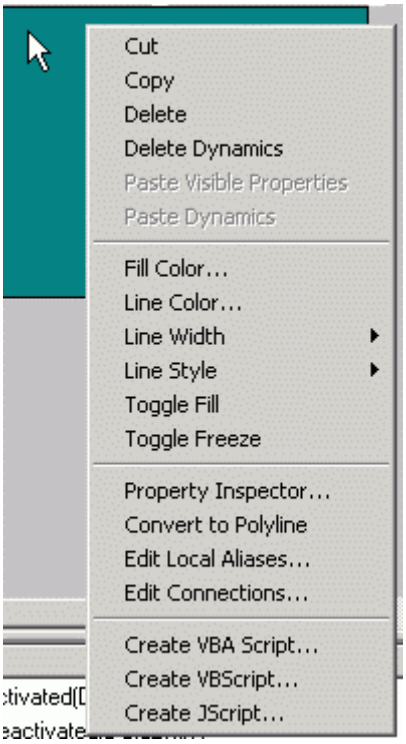
1. Click the **Selector** button on the **Draw** toolbar or right-click the work area to access the **Selector** tool from a list of options. The mouse pointer appears as an arrow.
2. Select an object by placing the pointer on the object and clicking the left mouse button. Square handles surround the object.

To select multiple objects:

1. Click the **Selector** button on the **Draw** toolbar.
2. Draw a box to surround all the objects by pressing the left mouse button and dragging the pointer around the objects, or by pressing the **SHIFT** key and clicking the individual objects.
3. Release the mouse button. Square handles surround each object selected.

Note: The **Select** mode is always selected by default on the **Draw** menu.

Right-clicking the selected object opens a special version of the **Format** menu, as shown below, which is discussed later in this section.



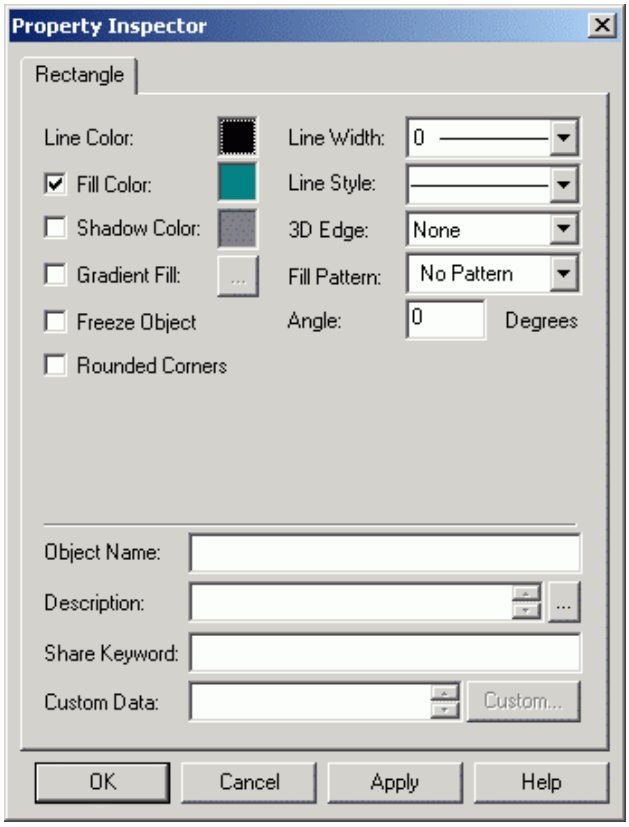
Format Menu

Property Inspector

The **Property Inspector,** shown below, is a property sheet containing a collection of property pages, or tabs. The Property Inspector dialog box lets you view and make changes to object parameters. The primary page is the General Page, which reflects the properties of a given static object (in the figure shown below, a rectangle).

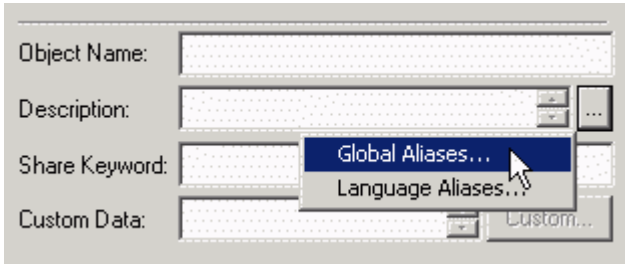
The Property Inspector may also display a variable number of additional pages that show information about attached dynamics. These additional pages are easily accessed via the tabs at the top of the dialog box.

The Property Inspector has four buttons on its Property Sheet: **OK, Cancel, Apply,** and **Help.** The **OK** button accepts all changes made on all of the property pages and exits the dialog. The **Cancel** button discards all changes made on all of the property pages and exits the dialog box. The **Apply** button accepts all changes made on all of the property pages but does not exit the dialog (subsequently canceling from the Property Inspector will not undo applied changes since those changes have already been accepted).



Property Inspector Dialog Box

In the **Description** field, you can select an alias to use by clicking the **...** button to the right of the text box and selecting either **Global Aliases** or **Language Aliases** from the pop-up menu, as shown in the figure below. This opens the respective alias browsers, allowing you to choose from a list of available aliases.



Selecting an Alias for the Description Field

Checking and Altering the Parameters of an Object

Lines

Segmented Lines

Rectangles

Ellipses

Text

Arcs

Checking and Altering the Parameters of an Object

To check and alter the parameters of an object:

1. Double-click the object. This opens the **Property Inspector** dialog box.
2. Edit the parameters described in the table below, and then click **OK** to return to the work area.

Note: The fields of the **Property Inspector** dialog box vary slightly depending on what type of object is being inspected.

Property Inspector Parameters

Parameter	Description
Line Color	Current line color of the object. Click this field to display the Color Palette and select a line color from the options available.
Fill Color	Current fill color of the object. Click this field to display the Color Palette and select another color from the options available.
Shadow Color	Current shadow color. Click this field to display the Color Palette and select a shadow color for the object from the options available.
Gradient Fill	Clicking the '…' button allows you to set the Gradient Fill options for the object.
Freeze Object	Checking or unchecking the box enables you to enable or disable the ability to move or stretch this object.
Rounded Corners	Rounds out the corners of a rectangle.
Line Width	Current Line Width. Change the width by selecting from the list box.
Line Style	Current Line Style of the object. Change the style by selecting from the list box.
3D Edge	Gives the selected object the 3D edge specified from the options available.
Fill Pattern	Fills the selected object with the chosen pattern from the available options.
Angle	Specifies the angular orientation of the object.
Object Name	Identifies the object for OLE Automation.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the **...** button to the right of the Description field.
Share Keyword	Used to identify the object for Update Shared Objects.
Custom Data	Allows you to enter custom data for the object.

Gradient Fill

The **Gradient Fill** feature in the Property Inspector is supported for display background, rectangles/rounded-rectangles, ellipses, pies/chords, lines and text - back - fill (including process points, buttons and times/dates). To access the gradient feature, you must first insert one of the previously mentioned items in the display and then double-click the item to open the Property Inspector.

Gradient Configuration Features
Adding Dynamic Actions
Gradient Styles
Offsets
Brightness
Detail Steps
Quick Set Buttons

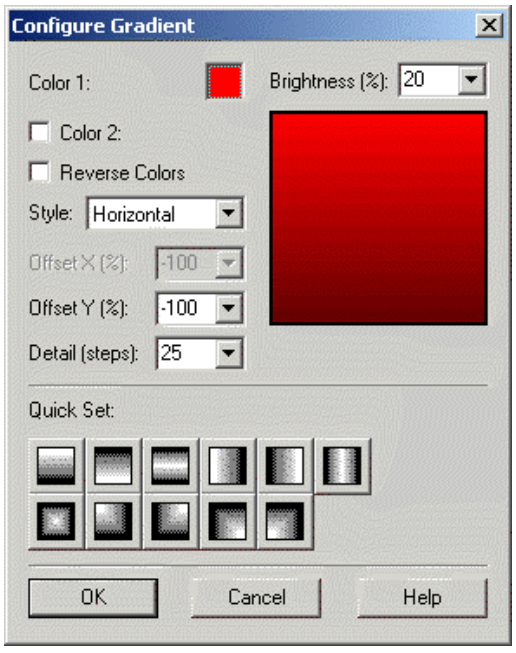
Gradient Configuration Features

It is possible to change the color of the object from the **Configure Gradient** dialog box, shown below. To do so, check the **Gradient Fill** check box in the Property Inspector and then click the **...** button. Click on the color box in the **Configure Gradient** dialog box to open the **Color Palette.**

It is also possible to reverse the color (when using both single and two color gradients) by checking the **Reverse Colors** check box.

Note: The color of the gradient object can be changed using the Color Palette.

Additionally, the **Configure Gradient** dialog box contains a preview window that shows what the gradient fill looks like as you adjust the gradient properties.



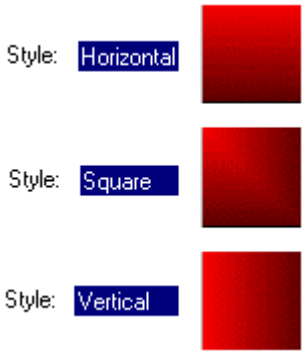
Configuring Gradients

Adding Dynamic Actions

When you add a dynamic action to an object that has a gradient, the object acts as it would normally, but the coloration now contains the gradient fill. However, notice that if the object is given a rotation dynamic, the shading of the gradient does not stay in the same place at all times (i.e., as the object rotates past the *x*-axis, the shading flips sides of the object).

Gradient Styles

There are three different gradient styles: **Horizontal, Square** and **Vertical.**



The Horizontal style has the light focus coming from the top or the bottom of the object.

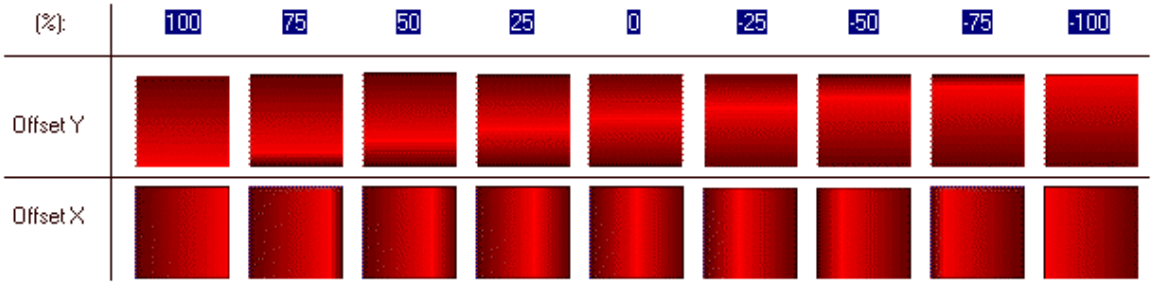
The Square style has the light focus coming from a corner.

The Vertical style has the light focus coming from either the right or the left side of the object.

Gradient Styles

Offsets

The **Offset** feature in the **Configure Gradient** dialog box sets how far off center the light focus is from both the *x*-axis and *y*-axis. Two options are available: **Offset X** and **Offset Y**.



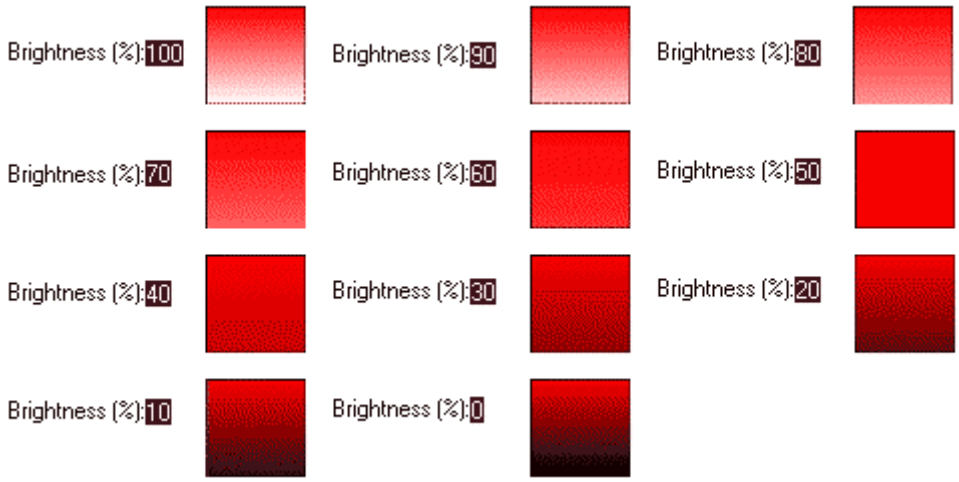
Gradient X and Y Offsets by Percentage

The figure shown above is an example of nine different light sources for both the Offset X and the Offset Y.

Note: It is possible to enter any number between 100 and -100 for both the X and Y offsets.

Brightness

If you are only using one color, you can select a percentage of brightness. The following is an example of some of the different percentages of brightness that are possible through the **Configure Gradient** dialog box. You can enter any number between 0 and 100 for the brightness level.

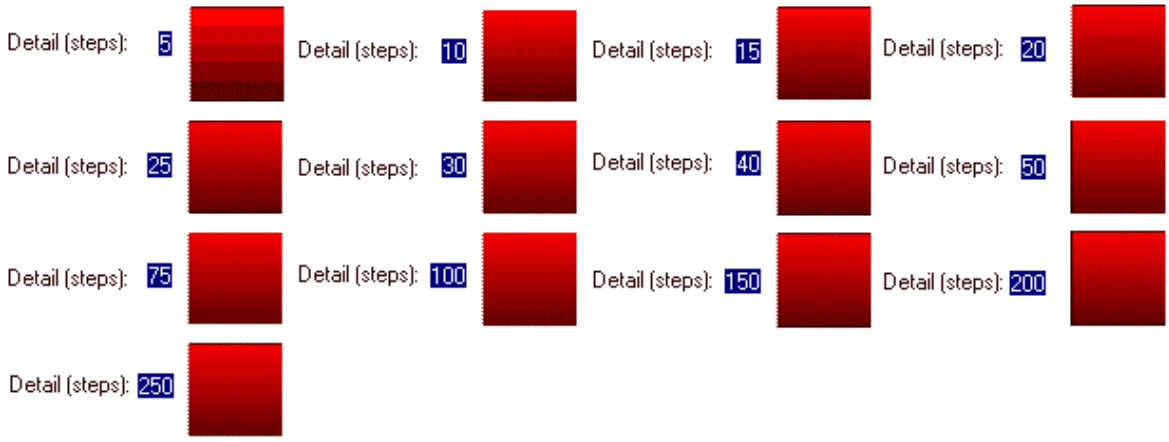


Gradient Brightness by Percentage

If you are using two colors, the brightness adjustment option disappears and you are instead allowed to select the second color. This second color is tied to the line color of the shape, even if the second color option is not selected. For example, if you change the second color to yellow and then decide to switch back to a one-color gradient, the borderline of the object will still be yellow.

Detail Steps

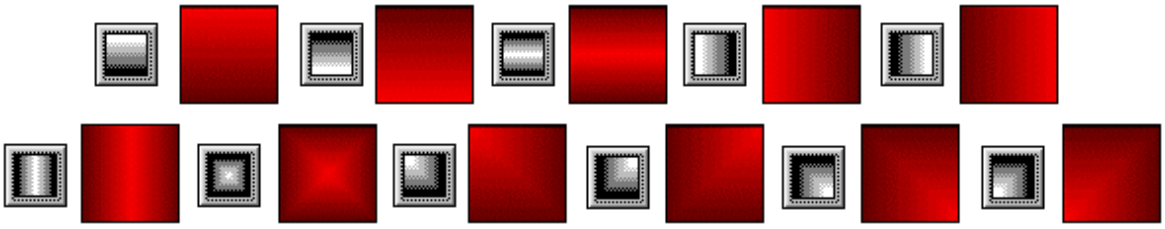
The **Detail (Steps)** feature in the **Configure Gradient** dialog box sets the smoothness of the gradient's appearance. The following is an example of how some of the different detail steps appear.



Detail Steps for Gradients

Quick Set Buttons

The **Configure Gradient** dialog box also contains the following **Quick Set** buttons. Each quick set button has a predefined configuration, which will override any settings you have already defined for a particular object.



Gradient Quick Set Buttons

Lines

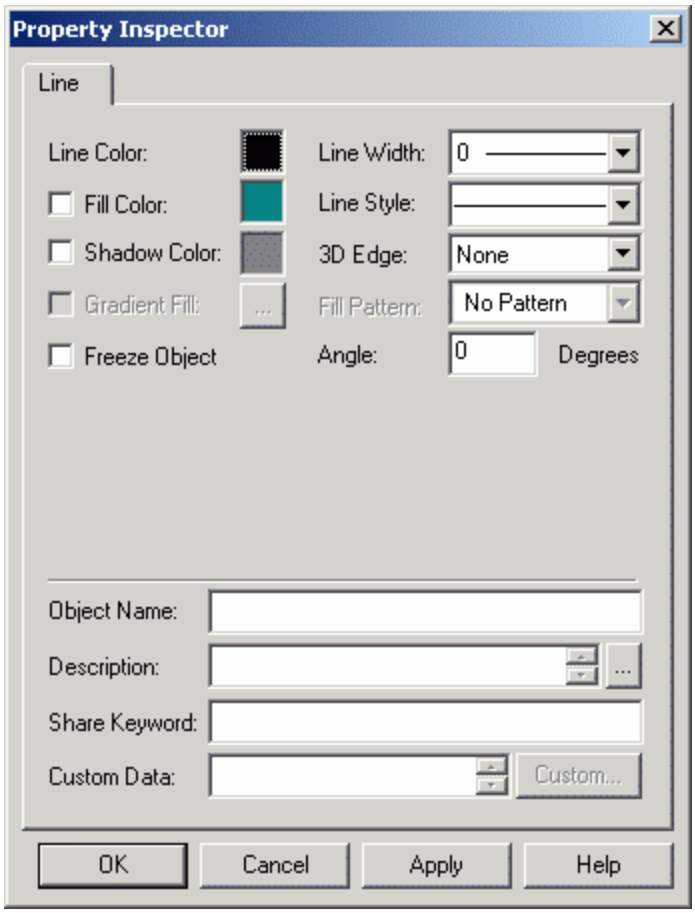


The **Line** tool on the **Draw** toolbar enables you to draw straight lines in the work area. You can also define the line style (solid or segmented) and line weight (thickness) using the **Line Style** and **Line Width** functions. Refer to the Formatting Objects section of this topic for more information.

To draw a line:

1. Select **Straight Line** from the **Draw** menu, or click the **Straight Line** button on the **Draw** toolbar. The mouse pointer appears as a pencil.
2. Pressing **SHIFT** while you draw constrains the line to 0, 45 or 90 degrees.

Double-clicking on the line displays the Line Property Inspector, shown below. Right-clicking the line displays the **Format** menu.



Property Inspector: Line

To alter the length of a line:

1. Select the line.
2. Left-click a square handle of the select box, drag it to extend the line, and release.

To reposition a line:

1. Click the line.
2. Drag it to a new position.
3. Release the mouse button.

To edit the vertices of a line:

1. Select the line.
2. Press **SHIFT** and right-click.
3. The line's vertices appear, and the mouse cursor appears as an upward-pointing arrow. You can now edit the midpoints and vertices of the segments.

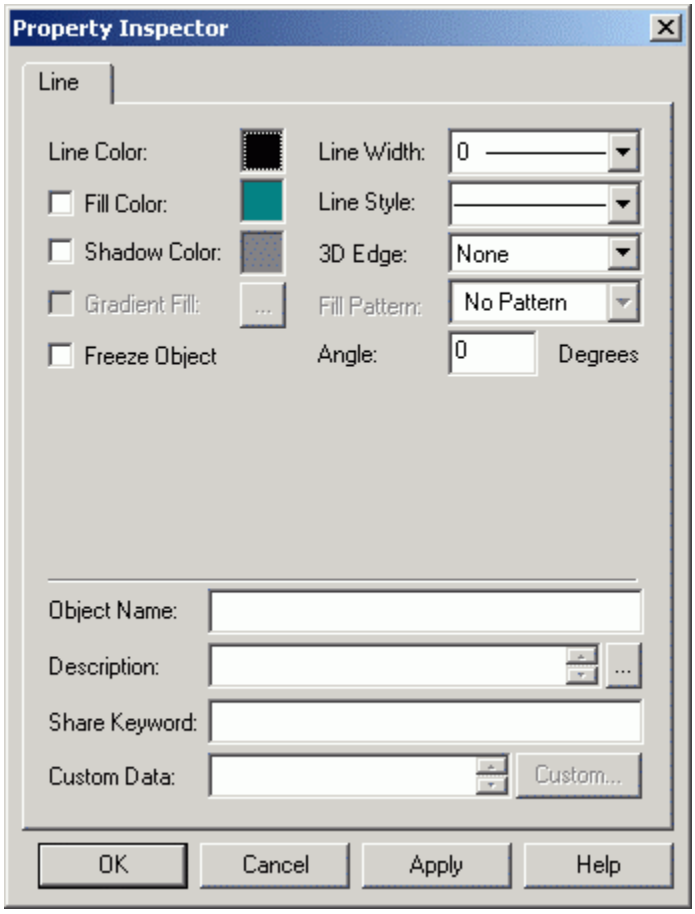
Segmented Lines



The **Segmented Line** button on the **Draw** toolbar allows you to draw a flexible, segmented line that can be straight or curved.

Keep the left mouse key pressed for free-hand drawing. Click and release the mouse from point to point. Double-clicking on the line opens the Property Inspector specific to the segmented line, which is identical to the Property Inspector page for the straight line.

When you create a segmented line, and the final vertex is close to the starting vertex, the polygon will automatically be closed off.



Property Inspector: Line

Rectangles

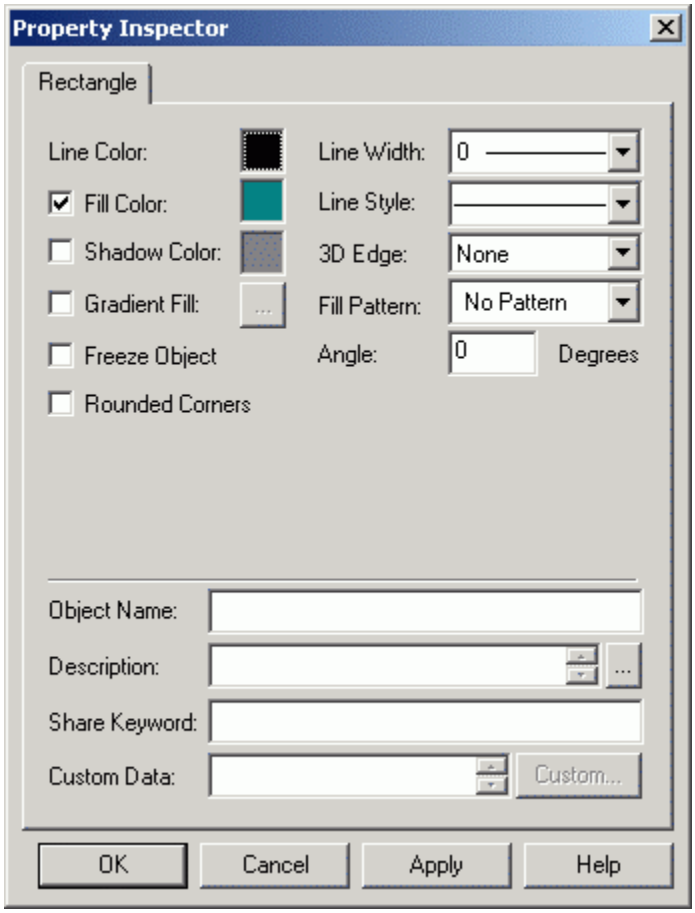


You can use the **Rectangle** tool to draw filled or unfilled rectangles in the work area. The color of the object's border is the currently selected color in the Color Palette.

To draw a rectangle:

1. Select **Rectangle/Square** from the **Draw** menu, or click the **Rectangle/Square** button on the **Draw** toolbar. The mouse pointer changes to a box and cross.
2. Click the left mouse button and drag to form the box. To form a square box, press **SHIFT** while you draw the box.

To view the properties of the box, double-click on the box. The Property Inspector for the rectangle object opens, as shown in the figure below. The Rectangle properties contain the unique check box **Rounded Corners,** which rounds out the corners of the rectangle or square, in addition to the standard property parameters.



Property Inspector: Rectangle

Right-clicking on the rectangle displays the **Format** menu, which provides options for formatting the rectangle. The **Format** menu is discussed later in this section.

Note: The cursor remains in the rectangle mode until you double-click, right-click, or press **Esc.** This applies to all drawing operations.

To resize a rectangle:

1. Select the rectangle, and then click the left mouse button on a square handle; drag to resize.
2. Use the **SHIFT** key to maintain the aspect ratio of the rectangle while resizing.

Filling and Unfilling

You can fill and unfill color in rectangles by using the toggle fill function explained in the Formatting Objects section in this topic. You can also change the color by selecting the rectangle and selecting a different color from the **Color Palette.**

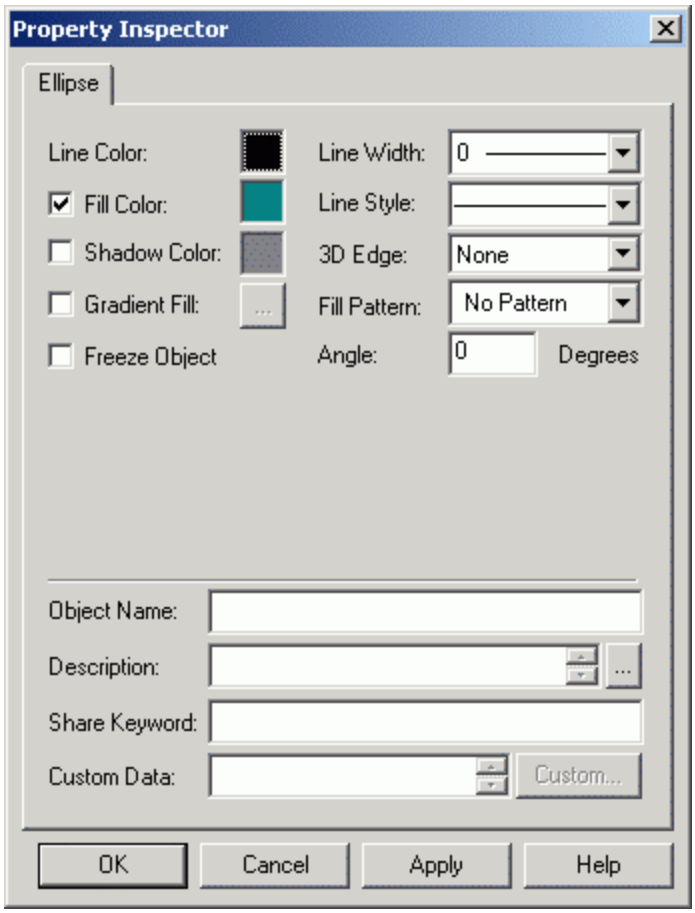
Ellipses



With the **Ellipse** tool, you can draw elliptical or circular objects.

To draw an ellipse:

1. Select **Ellipse/Circle** from the **Draw** menu, or click the **Ellipse** in the **Draw** toolbar.
2. Click the left mouse button and drag the pointer anywhere in the work area to form an ellipse. To form a circle, press **SHIFT** while you draw.



Property Inspector: Ellipse

Right-clicking the ellipse displays the **Format** menu.

To resize the ellipse:

1. Select the ellipse.
2. Click the left mouse button on a square handle and drag it to resize the ellipse. Press **SHIFT** while resizing to maintain the aspect ratio of the object.

Text



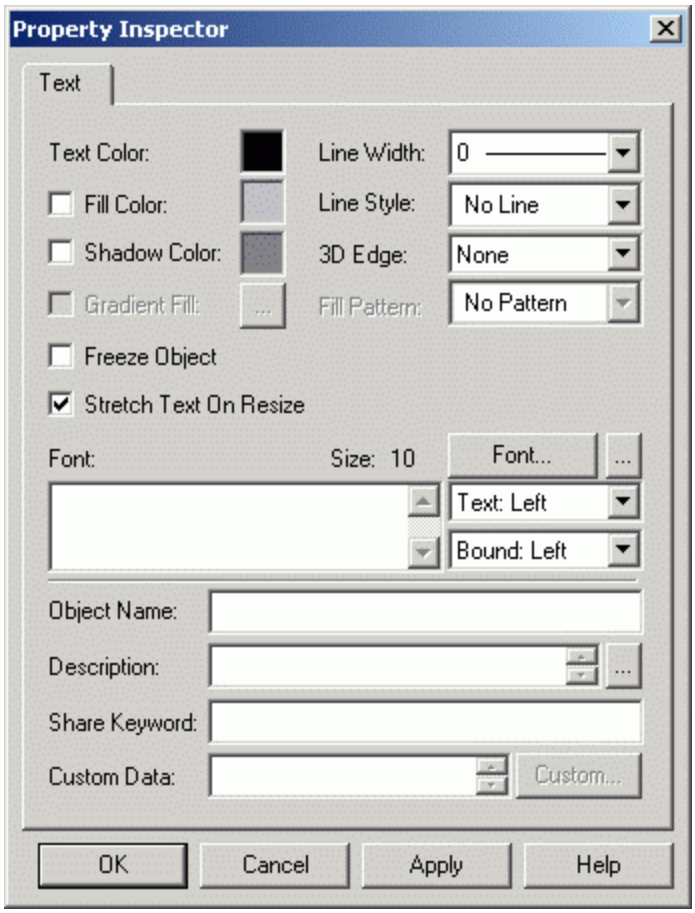
Use the **Text** tool to insert text into the work area.

To insert text:

1. Select **Text** from the **Draw** menu, or click the **Text** button on the **Draw** toolbar.
2. Left-click on the work area where you want to place the text object.
3. Type the text you wish to appear and left-click outside the text to insert.

When you right-click on the selected text object, the **Format** menu appears with the **Edit Text** option. Right-click and press **SHIFT** to directly edit the text on

screen. Or you can double-click on the object to open the Property Inspector, and then edit the text attributes according to the parameters given in the dialog box, as shown below.



Property Inspector: Text Object

You can type a text message directly inside the box under **Font.** You can select an alias to use for the text message by clicking the **...** button to the right of the Font button and selecting either **Global Aliases** or **Language Aliases** from the pop-up menu, as shown in the figure below. This opens the respective alias browsers, allowing you to choose from a list of available aliases.



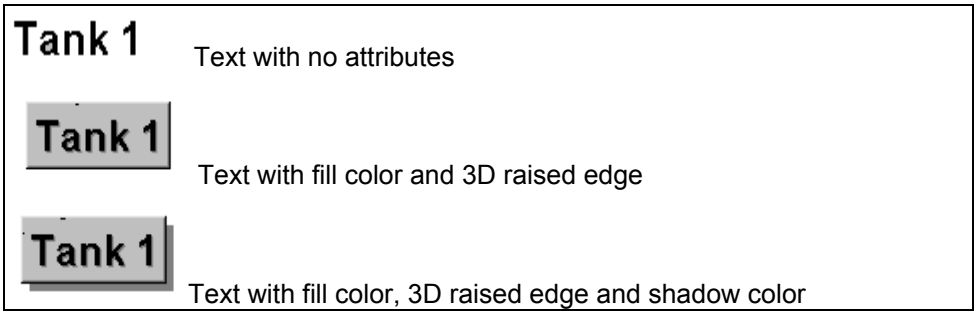
Selecting an Alias for the Text Box

The text object Property Inspector parameters are listed in the table below:

Text Property Inspector Parameters

Parameter	Description
Text Color	Specifies a text color from the Color Palette.
Fill Color	Specifies the color of the text background from the Color Palette.
Shadow Color	Specifies the color of the text shadow from the Color Palette.
Gradient Fill	If the **Fill Color** box is checked, the fill color will have a gradient applied to it.
Freeze Object	Locks the text object in one position.
Stretch Text on Resize	If this option is checked, the text will stretch with the text box when it is resized.
Line Width	Determines the width of the text box.
Line Style	Determines the style of the text box.
3D Edge	Determines the style of the 3D edge.
Fill Pattern	Determines a pattern for the background of the text object.
Font Selection	
Text Entry Area	This allows you to edit the text in the text box.
Font	Displays the font used for the selected text.
Size	Displays the point size for the selected text.
Style	Displays style of the selected text.
Font button	Opens the font dialog box to change the parameters for selected text object, such as font, size, and style.
Text: Left, Center, or Right	Aligns the paragraph text to the left, center, or right.
Bound: Left, Center, or Right	Bounds text to the left, center, or right of text box.

The figure shown below provides some examples of how you can configure text to appear in your display:



Adding Text Attributes

Arcs

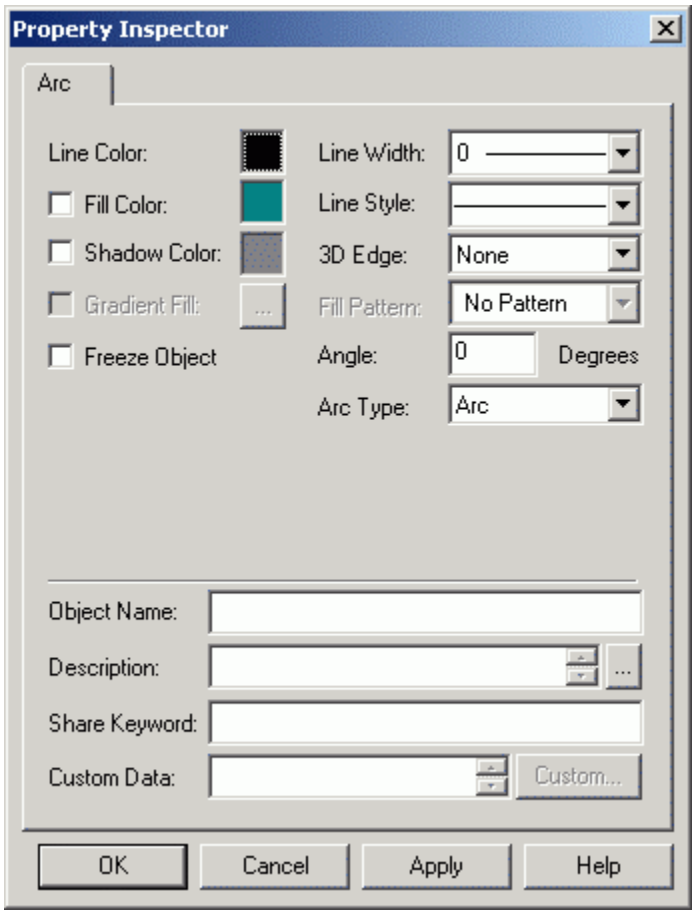


You can use the **Arc** tool to draw arc-shaped objects in the work area.

To draw an arc:

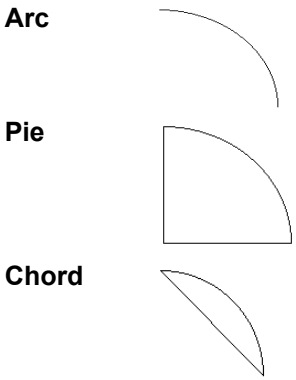
1. Select **Arc** from the **Draw** menu, or click the **Arc** button on the **Draw** toolbar.
2. To determine the arc's radius, press and hold the left mouse button, drag the pointer away from the start point, and click.
3. To determine the arc's length, press and hold the left mouse button over one of the boxes on either end of the arc. Drag the arc to the desired length.

To view the properties of the arc, double-click it to bring up the Arc Property Inspector, shown below.



Property Inspector: Arc

In addition to the standard parameters, this Property Inspector contains an **Arc Type** selection. The figures below show examples of arc types.



To resize an arc:

1. Select the arc.
2. Press the left mouse button on a square handle, and then drag it to resize the arc and release.

To edit the length of an arc:

1. Right-click the arc to display the **Format** menu, and then choose **Edit Arc.**
2. The mouse pointer changes to a box and cross, allowing you to change the length of the arc.

To convert an arc to a polyline:

1. Select the arc.
2. Right-click the arc to display the **Format** menu, and then choose **Convert to Polyline.**
3. Now you have the option of merging the arc with other line objects to create a single polyline object. This allows you to apply properties such as fill to the whole object. To merge two objects, select both objects and then select **Group Into Symbol** from the **Arrange** menu or toolbar.

Import Functions

Images
Metafiles
Symbols

Images



By clicking the **Import Image** button on the **Draw** toolbar, or selecting **Import > Image** from the **Draw** menu, you can import an image file into your work area. Images can be resized and moved within a GraphWorX32 display.

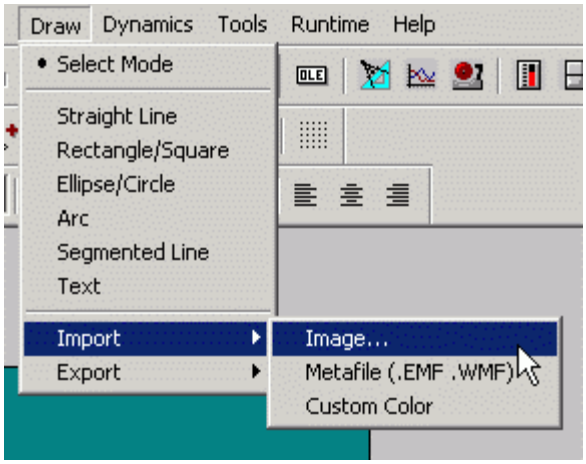
Since the early versions, GraphWorX32 has been able to import external image files into display files. Until now the import functionality has been limited to the only bitmap (.bmp) image file format. The bitmap format has been in the past years the most common over any Microsoft platform, but today many other graphic formats are available and widely used.

Unfortunately the bitmap format is also the simplest one, and the least efficient in terms of disk space. Some graphic formats (e.g. .jpg and .png) are used because of the minimum disk space that they require, while other formats (e.g. Adobe Photoshop .psd) are used for their flexibility and powerful capability. GraphWorX32 supports the following image types for import:

- Adobe Photoshop Files (*.PSD)
- C64 Koala Graphics (*.KOA)
- CompuServe GIF (*.GIF)
- Dr. Halo (*.CUT)
- Icon (*.ICO)
- IFF Interleaved Bitmap (*.IFF)
- JPEG (*.JPG;*.JPEG)
- Kodak PhotoCD (*.PCD)
- Liner Bitmap Graphics (*.LBM)
- Multiple Network Graphics (*.MNG)
- Zsoft Paintbrush (*.PCX)
- Portable Network Graphics (*.PNG)
- Portable Network Media (*.PBM;*.PGM;*.PPM)
- Sun Raster Images (*.RAS)
- Truevision Targa (*.TGA;*.TARGA)
- Tag Image File Format (*.TIF;*.TIFF)
- Windows or OS/2 Bitmap (*.BMP)
- Wireless Bitmap (*.WBMP)

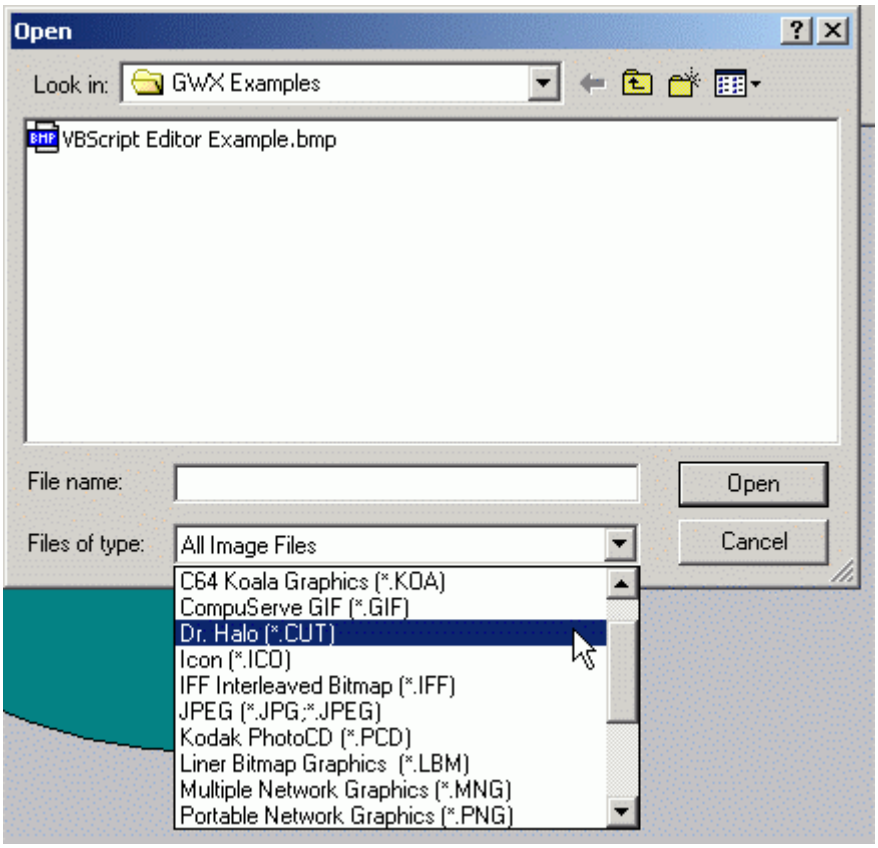
To import an image into a GraphWorX32 display file:

1. Select **Import > Image** from the **Draw** menu in GraphWorX32 (or click the **Import Image** button on the **Draw** toolbar), as shown in the figure below.



Importing an Image File Into a GraphWorX32 Display File

2. In the **Open** dialog box, select an image file type from the **Files of type** drop-down list, as shown in the figure below. Select an image file to import, and then click **Open.**
3. The image is inserted into the GraphWorX32 display.

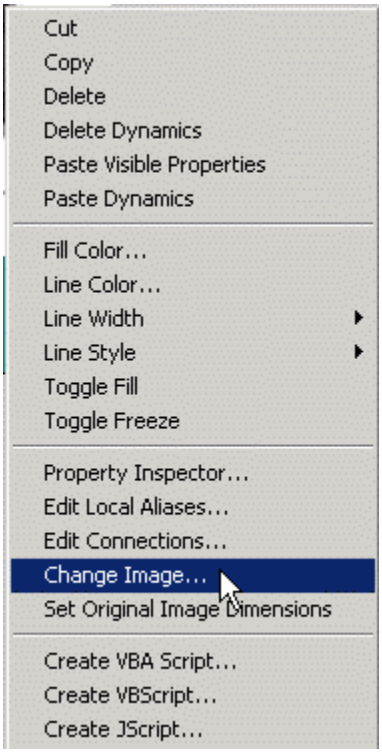


Selecting an Image File and Image Type To Import

Changing an Image File

To change an image that has been imported into a GraphWorX32 display:

1. Right-click the image and choose **Change Image** from the pop-up menu.
2. In the **Open** dialog box, select an image file type from the **Files of type** drop-down list. Select an image file to import, and then click **Open.**



Changing an Image

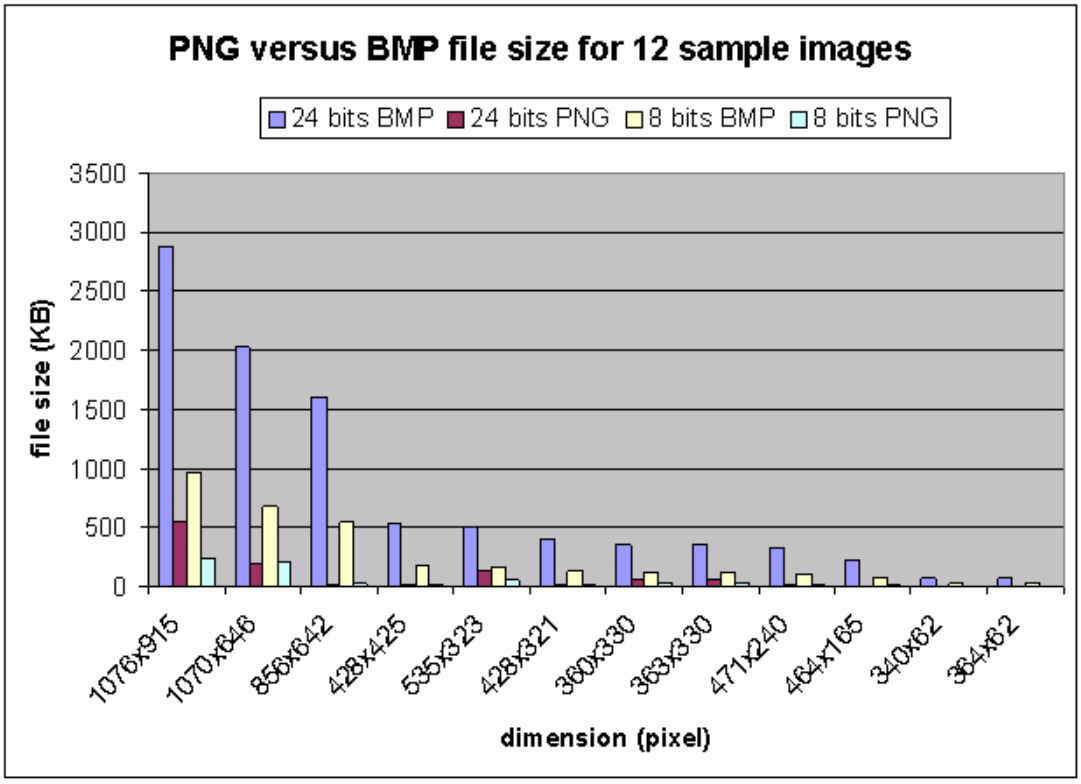
Neural Network / PNG Image Compression

GraphWorX is capable of importing many images formats. Some of these formats use compression (i.e. .jpeg) and others do not (i.e. .bmp). Large images may require a lot of disk space: For example, a single true color bitmap file 1024 X 768 pixels may require up to 2 MB of disk space.

In order to reduce the disk space required from GraphWorX32 displays, a compression algorithm has been added. The image compression can help save disk space and optimize performance when manipulating bitmaps or delivering displays over the Internet through WebHMI.

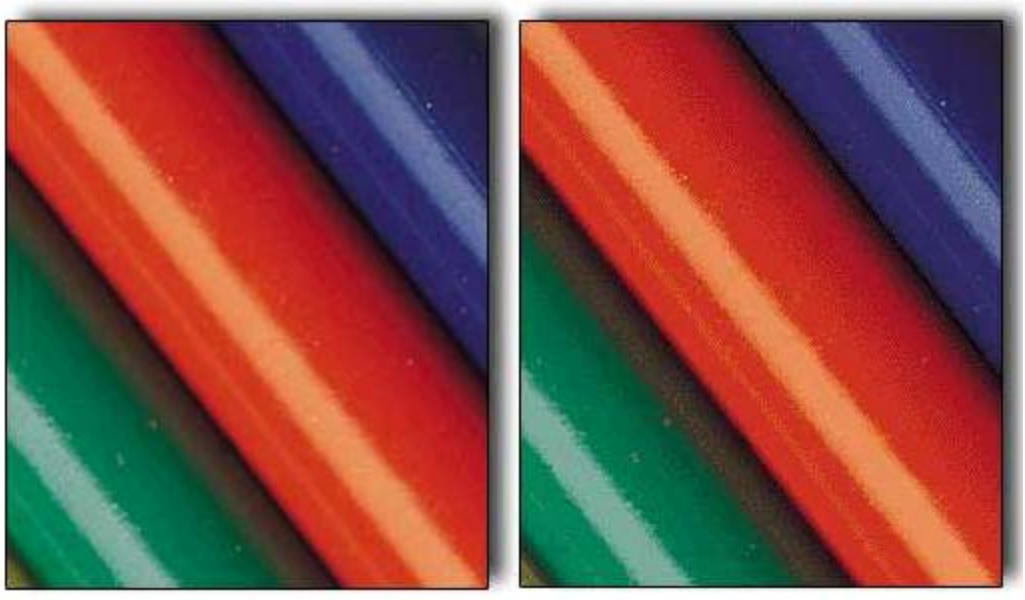
The image compression algorithm is based on Neural Network technology and PNG compression. The Neural Network helps reduce the color needed to represent an image, thus reducing the final file size. The PNG compression algorithm saves additional disk space.

The following comparison chart illustrates the differences in file size between standard BMP storage and advanced PNG compression.



Comparison of BMP and PNG Image Formats

The images below show the Neural Network color reduction algorithm. The image on the left uses 16 million colors to represent shades of red green and blue. The image on the right is the result of a first pass trough the Neural Network color optimization algorithm. The two images look almost identical, and only a sharp eye can tell the differences between the two, mainly related to the dithering.



Original Image (16 million colors) **Color Reduced Image (256 colors)**

Metafiles



By selecting the **Import Metafile** button on the **Draw** toolbar, or by selecting **Import > Metafile** from the **Draw** menu, you can import a windows metafile (.wmf) or an enhanced metafile (.emf) into your work area. As with other images, it is possible to resize and move a metafile within a GraphWorX32 display.

Custom Colors

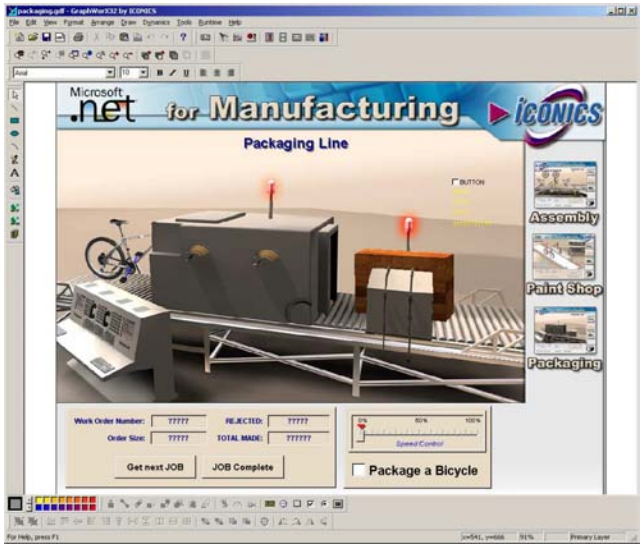
By selecting **Import > Custom Color** from the **Draw** menu, you can import a text file that contains red, green, and blue color parameters for the selected object in the display.

Export Functions

Exporting Images and Metafiles

By selecting **Export** from the **Draw** menu, you can export an enhanced metafile (.emf) or an image from your work area.

Sometimes it is useful to have an image that represents a GraphWorX32 display. GraphWorX32 provides a tool for exporting a GraphWorX display into an image file. The **Export > Image File** command on the **Draw** menu is available only if something is present inside the display. The resulting image file represents the display content as it is seen on the monitor. The windows and all the toolbars are stripped out of the image. For example, if the original display before export looks like this...



Display File Before Export

…then the final exported image file looks like this:

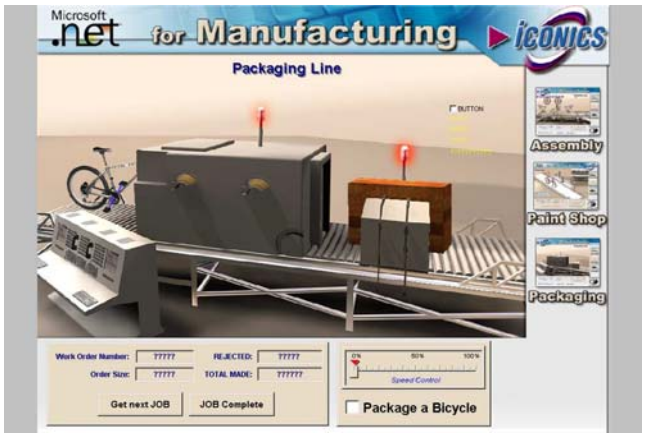


Image File After Export

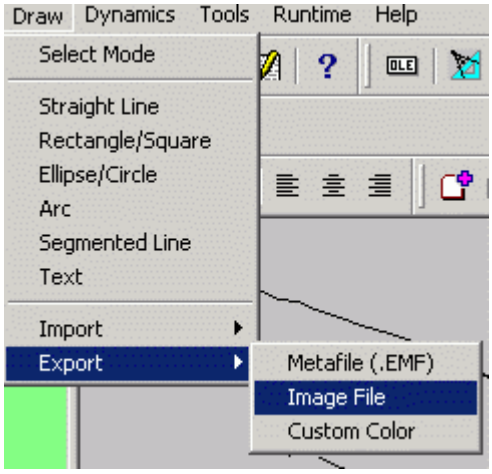
The following image types are supported for export:

- Bitmap (*.bmp)
- Jpeg (*.jpg)
- Png (*.png)
- Tiff (*.tif)

Each format has advantages and drawbacks. The bitmap format, for example, requires more disk space. The jpeg format saves some disk space but also sacrifices some image quality.

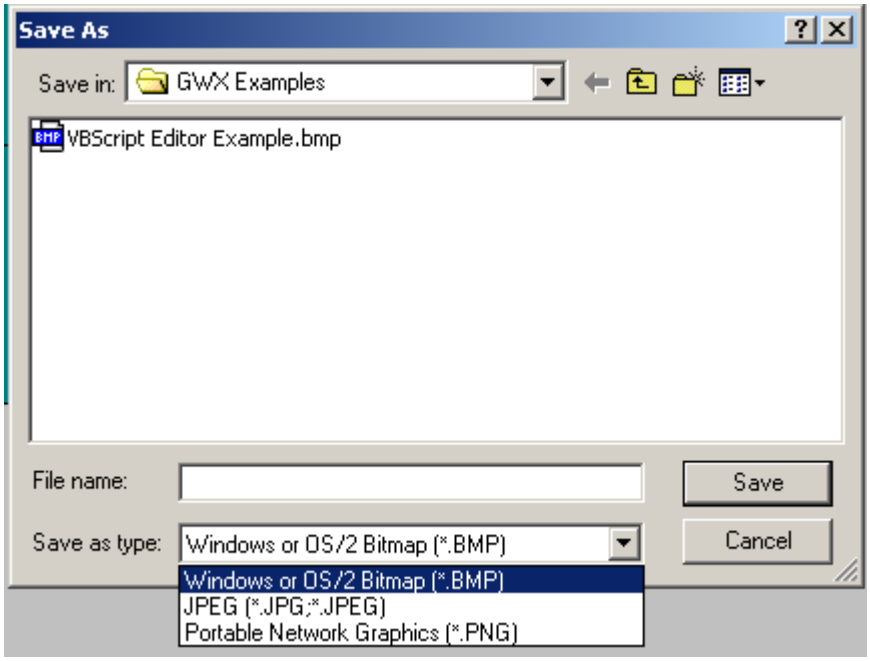
To export a GraphWorX32 display file to an image file:

1. Select **Export > Image File** from the **Draw** menu in GraphWorX32, as shown in the figure below.



Exporting a GraphWorX32 Display File to an Image File

2. In the **Save As** dialog box, select an image file type from the **Save as type** drop-down list, as shown in the figure below. Give the image file a name, and then click **Save.**



Saving the Image File

Exporting Custom Colors

By selecting **Export > Custom Color** from the **Draw** menu, you can export a text file that contains red, green, and blue color parameters for the selected object in the display.

Symbols

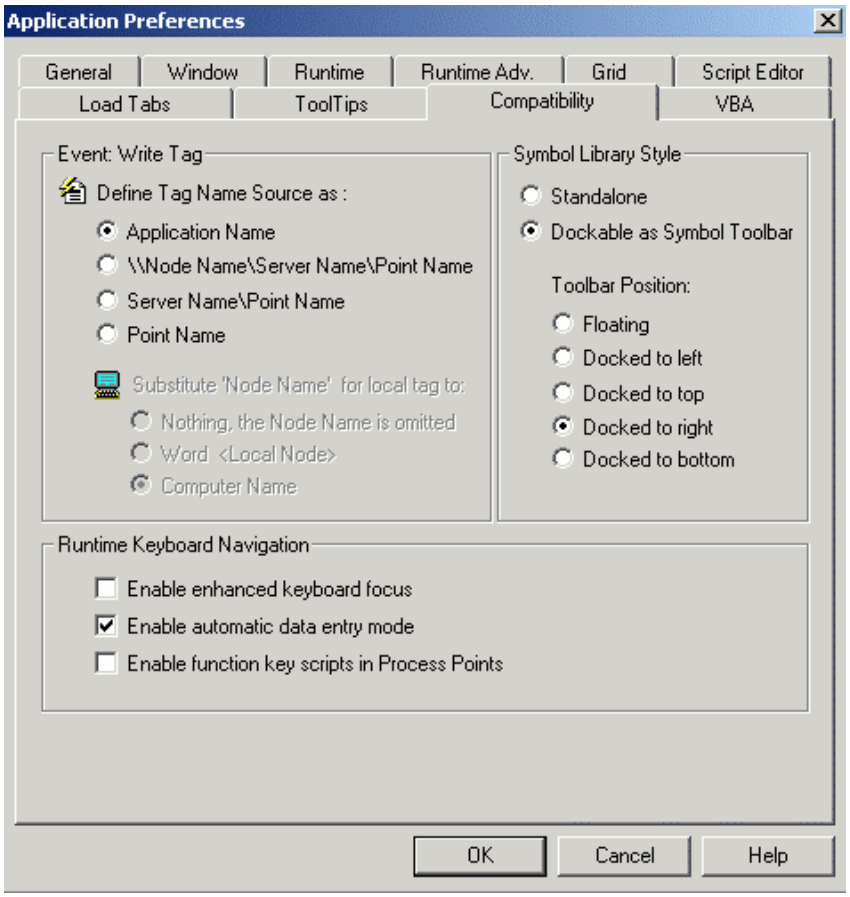


GraphWorX32 uses two different modes for the Symbol Library:

- Standard stand-alone Symbol Library
- Dockable symbol toolbar

To switch between these two modes:

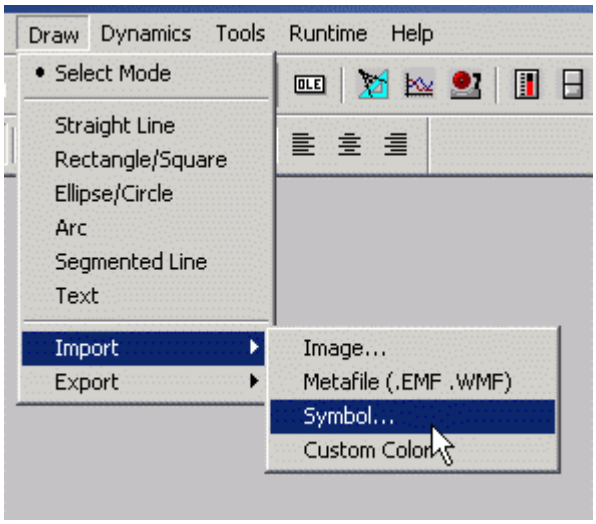
1. Select **Application Preferences** from the **Format** menu.
2. The **Application Preferences** dialog box appears, as shown in the figure below. Select the **Compatibility** tab. Under the **Symbol Library Style** field, you can select **Stand-alone** mode or **Dockable as Symbol Toolbar** mode.



Switching Between Stand-alone and Dockable Symbol Modes

Stand-alone Symbol Library

If **Stand-alone** mode is active, clicking the **Symbols** button on the **Draw** toolbar launches the **SymbolLibrary.exe.** You can also select **Import > Symbol** from the **Draw** menu, as shown in the figure below.



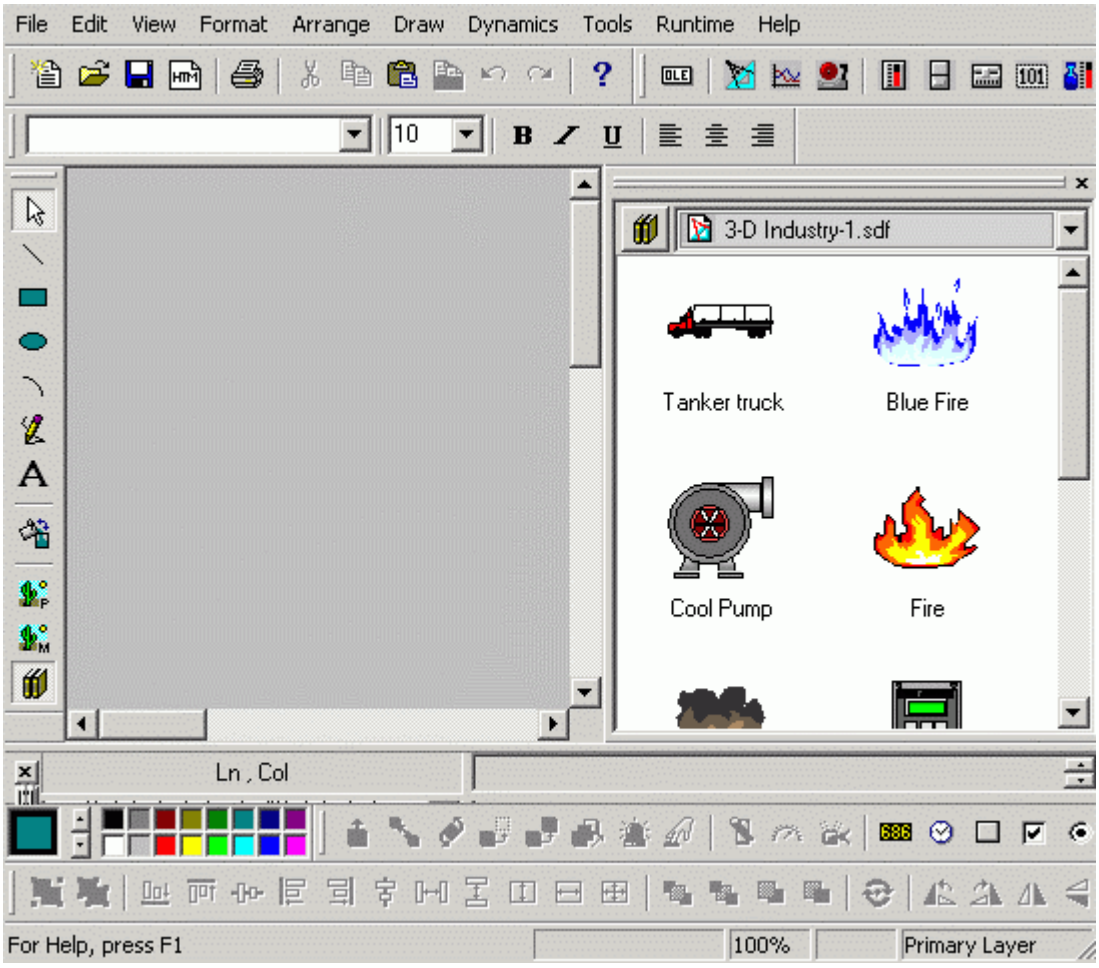
Importing Symbols

Dockable Symbol Toolbar

If the **Dockable as Symbol Toolbar** option is selected, then the Symbol Library is replaced with a Symbol toolbar inside the GraphWorX32 display, as shown in the figure below. The **Symbols** button on the **Draw** toolbar toggles the visibility of the symbol toolbar (instead of launching the Symbol Library). The **Toggle Symbol Toolbar** command on the View menu is enabled, and the **Show Toolbars** dialog (which appears when you select **Toolbars** from the **View** menu) also includes a **Symbols** check box for showing/hiding the symbol toolbar. The **Import > Symbol** command is removed from the **Draw** menu.

You can also position the symbol toolbar as:

- A floating window
- Docked to left
- Docked to right
- Docked to bottom
- Docked to top



Symbol Toolbar Docked Inside a GraphWorX32 Display

Registry Presettings

The mode for the Symbol Library can be specified during the installation process by creating the following registry entries under

[HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\Gwx32\Compatibility Properties]:

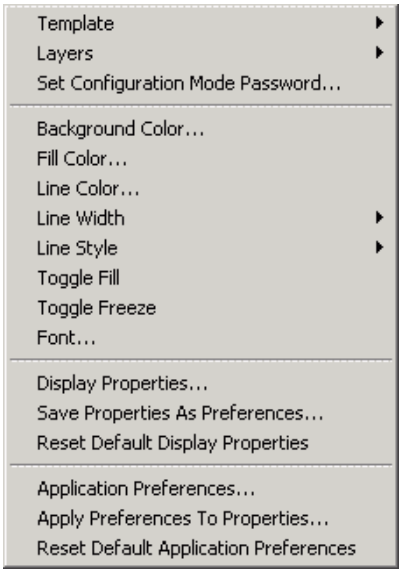
- **Compatibility Mode:** Can be either Stand-alone or Dockable Symbol Toolbar
- **Dockable Mode:** Symbol toolbar can be a floating window, docked to left, docked to right, docked to bottom, or docked to top

Note: These settings are saved for each user independently in

[HKEY_CURRENT_USER\Software\ICONICS\Gwx32\Compatibility Properties]

Formatting Objects

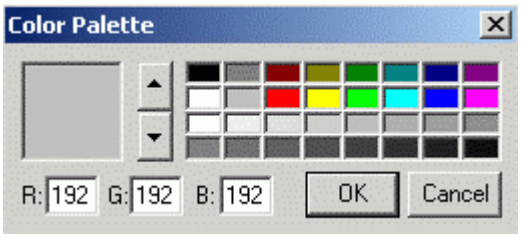
The **Format** menu provides options for layers, formatting line color, line style, line width, font type, and background color for your display. It also allows you to fill and unfill selected objects, to freeze and unfreeze objects, and to define font style and size.



Format Menu

Background Color

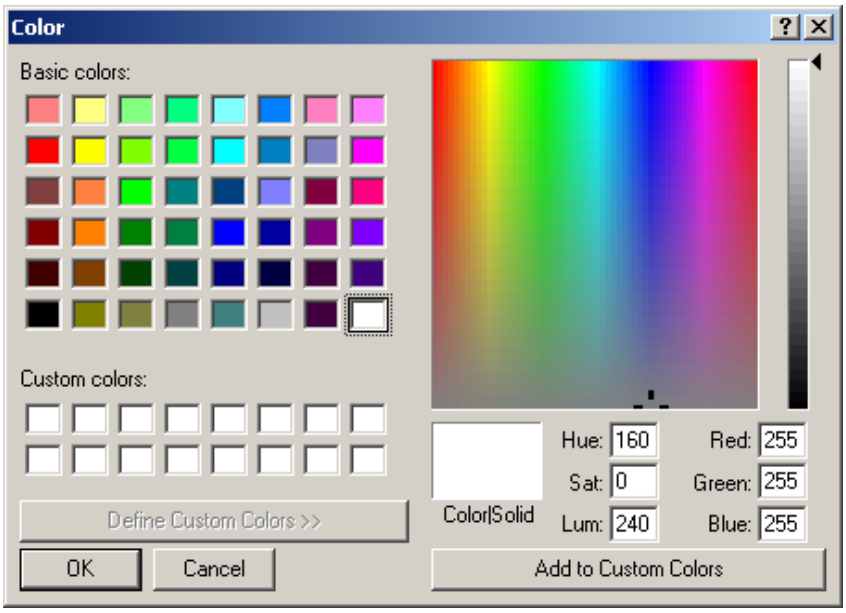
Selecting **Background Color** from the **Format** menu function defines the color of the background of the selected object from the **Color Palette,** shown below.



Color Palette

Color Palette Menu

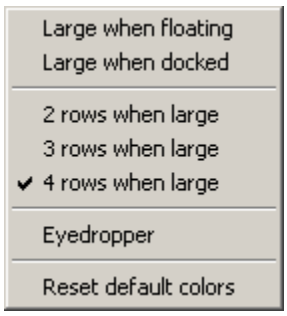
Double-clicking on the **Color Palette** displays the **Color** dialog box, shown below. This allows you to choose from a wider range of colors than are available on the basic color palette.



Color Dialog

You can use a slider to choose from a range of colors and patterns in the color spectrum at right. The color will be displayed in the **Color/Solid** box. You can then add a custom color or a textured color by clicking the **Add to Custom Colors** button.

Right-clicking the **Color Palette** toolbar displays the menu shown in the figure below. When any or all of these parameters are checked on the menu above, the color palette can be kept large when floating, large when docked, viewed with two rows when large, three rows when large, or four rows when large.



Color Palette Menu

Eyedropper

Selecting **Eyedropper** from the **Color Palette** menu allows you to select a color from anywhere in the display and apply it within the display. Click anywhere in the GraphWorX32 window over the color you want to pick up. Deposit the color in the color palette by clicking over the desired color box in the rows of available colors.

Reset Default Colors

You can restore the default colors by selecting **Reset Default Colors** from the **Color Palette** menu.

Fill Color

Selecting **Fill Color** from the **Format** menu opens the **Color Palette,** from which you can choose a color to fill a selected object in your display.

Line Color

Selecting **Line Color** from the **Format** menu opens the **Color Palette,** from which you can choose a color for a selected line in your display.

Line Style

Selecting **Line Style** from the **Format** menu defines how the line appears in a display. Lines can appear as solid or segmented. Currently this works with a line that has no width. There are several line styles from which to choose.

To define a line style:

1. Select an object (line, rectangle, segmented line, ellipse, or arc) in your display.
2. Select **Line Style** from the **Format** menu.
3. Select a line type.

Note: Line styles only exist for lines with a width of 0.

Line Width

Selecting **Line Width** from the **Format** menu defines the thickness of a line.

To define the line width:

1. Select an object (line, rectangle, segmented line, ellipse, or arc) in your display.
2. Select **Line Width** from the **Format** menu. The line width options appear to the right of the menu.
3. Select the line width.

Toggle Fill

Selecting **Toggle Fill** from the **Format** menu fills or unfills the inside of an object with the object's current color.

To fill an object with color:

1. Select the object.
2. Select **Toggle Fill** from the **Format** menu or click the **Toggle Fill** button on the **Draw** toolbar.

Note: Lines, images, and metafiles cannot be filled or unfilled with a color.

Toggle Freeze

Selecting **Toggle Freeze** from the **Format** menu disables the movement or stretching of aobjects in the work area. This function is useful for anchoring objects in a display.

To freeze or unfreeze an object:

1. Select the object.
2. Select **Toggle Freeze** from the **Format** menu. A lock appears on the object indicating it is frozen.

You can also freeze an object through the Property Inspector by checking the **Freeze** check box.

Font

Selecting **Font** from the **Format** menu defines the font types, styles, and point sizes for text. You can define font attributes before or after you create text objects.

To define font types:

1. Select the text object(s) in your display.
2. Select **Font** from the **Format** menu to open the **Font** dialog box.

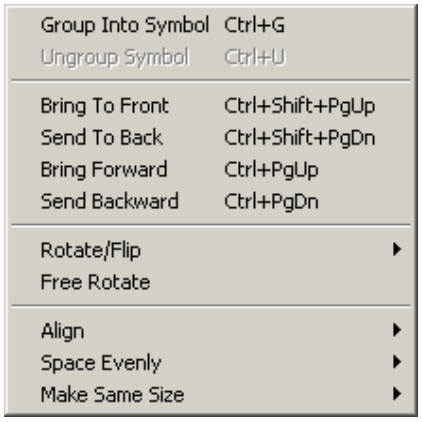
Define the font parameters and click **OK.** The text appears with the font parameters you defined.

Gradient Color

The color of the gradient object can be changed using the Color Palette toolbar.

Arranging Objects

The **Arrange** menu, shown below, and the **Arrange** toolbar provide you with options for moving objects in layers; grouping and ungrouping objects; rotating, aligning, and evenly spacing selected objects in the display; and making objects the same size.



Arrange Menu



Arrange Toolbar

Group Into Symbol



The **Group Into Symbol** function on the **Arrange** menu and the **Symbol** button on the **Arrange** toolbar groups several objects into one symbol. The resulting symbol can then be moved as one object.

Ungroup



The **Ungroup Symbol** function on the **Arrange** menu and the **Ungroup Symbol** button on the **Arrange** toolbar ungroup the symbol into its component objects.

Aligning Objects

The **Align** function on the **Arrange** menu enables you to align objects according to their bottoms, tops, middles, lefts, rights, and centers, as well as to make the objects the same height, the same width, and the same size. The **Align** function on the **Arrange** menu and the **Align** buttons on the **Arrange** toolbar adjust the relative positions of objects in the work area. You can align objects in the following positions:

Bottoms		Rights	
Lefts		Centers	
Middles		Even Across	
Tops		Even Down	

To align objects:

1. Select the objects.
2. Select **Align** from the **Arrange** menu, and choose a position or click on the appropriate button on the **Arrange** toolbar for align position.

You can make multiple objects in your display a uniform size, width, and height. This function applies to drawing objects (such as boxes, circles, and ellipses, push buttons, process points, time/date objects, text objects, sliders, imported graphics, and TrendWorX32 embedded windows.

On the **Arrange** menu, the **Make Same Size** function enables you to make all selected objects the same size (with respect to width, height, or both) as the object of comparison.

To make objects the same size:

1. Select the objects you wish to make a uniform size.

2. Each object that you select is surrounded by a box with white handles. Left-click the focus object. The selection box around the focus object now has blue handles.
3. Select **Make Same Size** from the **Arrange,** menu or click the **Make Same Size** button on the **Arrange** toolbar. All objects size to the focus object immediately.

Make Same Height 

Make Same Width 

Make Same Size 

Bring to Front



Select **Bring to Front** on the **Arrange** menu, or click the **Bring to Front** button on the **Arrange** toolbar, to move the selected objects in front of the objects associated with it in the current work area.

Send to Back



Select **Send to Back** on the **Arrange** menu, or click the **Send to Back** button on the **Arrange** toolbar, to move the selected objects behind all the objects in the current work area.

Bring Forward



Select **Bring Forward** on the **Arrange** menu, or click the **Bring Forward** button on the **Arrange** toolbar, to move the selected objects up one layer in the display.

Send Backward



Select **Send Backward** on the **Arrange** menu, or click the **Send Backward** button on the Arrange toolbar, to move the selected objects down one layer in the display.

Rotation

The **Rotation** function allows you to rotate a selected object freely, rotate left and right, and flip the object horizontally and vertically.

Free Rotation of Static Objects
Rotate Left and Right
Flip Horizontal/Vertical

Free Rotation of Static Objects



Free rotation of static objects means that the objects can be rotated to any angle. The following objects can be freely rotated:

- Polylines
- Rectangles
- Ellipses
- Arcs
- Symbols
- Images

Objects can be freely rotated through the **Angle** field of Property Inspector or through the **Free Rotate** button on the **Arrange** toolbar

Note: Objects that have been freely rotated can be flipped using the two flip buttons (see the **Horizontal/Vertical** Section).

Rotate Left and Right



Clicking the **Rotate Left** and **Rotate Right** buttons on the **Arrange** toolbar, shown above, allows you to rotate the selected object(s) to the left or the right in 90-degree increments. The following objects can be rotated by 90 degrees:

- Polylines
- Rectangles
- Ellipses
- Arcs
- Symbols
- Images
- Text

The ability to rotate text applies to the following:

- Multiline text
- Left, center, and right alignments
- Text objects
- Process points
- Data entries
- Statefields
- Timedates
- Buttons

Check boxes and radio buttons cannot be rotated.

Flip Horizontal/Vertical



Clicking on the **Flip Horizontal** and **Flip Vertical** buttons on the **Arrange** toolbar flips the selected object(s) on its horizontal or vertical axis. The following objects can be flipped:

- Polylines
- Rectangles
- Ellipses
- Arcs
- Symbols
- Images

Text and metafiles cannot be flipped.

Chapter 5: Display Edit Functions

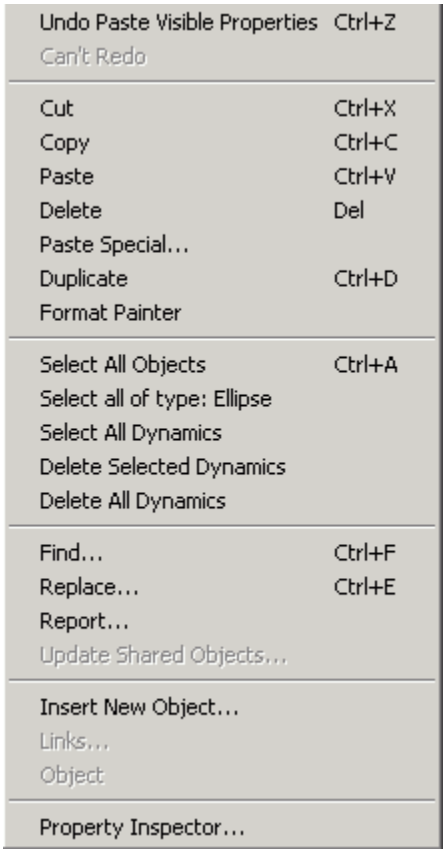
Contents

Introduction to Editing Functions
Undo/Redo
Basic Editing
Find, Report, and Replace
Wildcards
Insert Object
Update Shared Objects

Introduction to Editing Functions

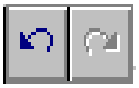
This section describes the GraphWorX32 **Edit** menu commands for altering objects in the display, including the following:

Undo	Select all of type
Cut	Select all dynamics
Copy	Delete dynamics
Paste	Find
Delete	Replace
Paste Special	Update shared objects
Duplicate	Insert new objects
Select all objects	



Edit Menu

Undo/Redo



Select **Undo** from the **Edit** menu to reverse the last editing action. Select **Redo** from the **Edit** menu to restore the last editing action that was undone. The name of the command changes depending on what the last action was. Select **Undo/Redo** from the **Edit** menu, or click the appropriate button on the **Main** toolbar. The **Undo** command changes to **Can't Undo** on the menu if you cannot reverse your last action.

Note: To undo several editing actions, continue to click the **Undo** button until all the edits you wish to remove have been undone. The same applies to the **Redo** command.

Shortcuts

Toolbar: 

Keys: **CTRL+Z** or **CTRL+Y**

Basic Editing

Basic editing functions available on the **Edit** menu include **Cut, Copy, Paste, Paste Special,** and **Duplicate.**

Copy



Select **Copy** from the **Edit** menu to copy selected data/object to the clipboard. This command is unavailable if no data/objects are currently selected.

To copy an object:

1. Select the object. A box with handles surrounds the object.
2. Select **Copy** from the **Edit** menu, or click the **Copy** button on the **Main** toolbar.

Copying data to the Clipboard replaces the contents previously stored there.

Shortcuts

Toolbar: 

Keys: **CTRL+C**

Cut



Select **Cut** from the **Edit** menu to move one or more selected objects to the windows Clipboard, removing them from the current display. Cutting data/objects to the clipboard replaces the contents previously stored there.

To cut an object:

1. Select the object to cut. A box with square handles surrounds the object.
2. Select **Cut** from the **Edit** menu, or click the **Cut** button on the **Main** toolbar. The object is removed from the display and sent to the Clipboard (here it remains until another object is placed onto the Clipboard.)

Note: You can also use the **DEL** key to cut selected objects and place them onto the Clipboard.

Shortcuts

Toolbar: 

Keys: **CTRL+X**

Paste



Select **Paste** from the **Edit** menu to place a copy of the objects currently in the Clipboard into the work area. The objects remain on the Clipboard until you copy or cut another object to the Clipboard. Use this command to insert a copy of the Clipboard contents at the insertion point. This command is unavailable if the Clipboard is empty.

To paste an object from the clipboard:

1. Select **Paste** from the **Edit** menu, or click the **Paste** button on the **Main** toolbar.
2. The object is pasted near the center of the GraphWorX32 display work area and is enclosed in a select box. Use the mouse pointer to move the object to another place in the work area if desired.

Shortcuts

Toolbar: 

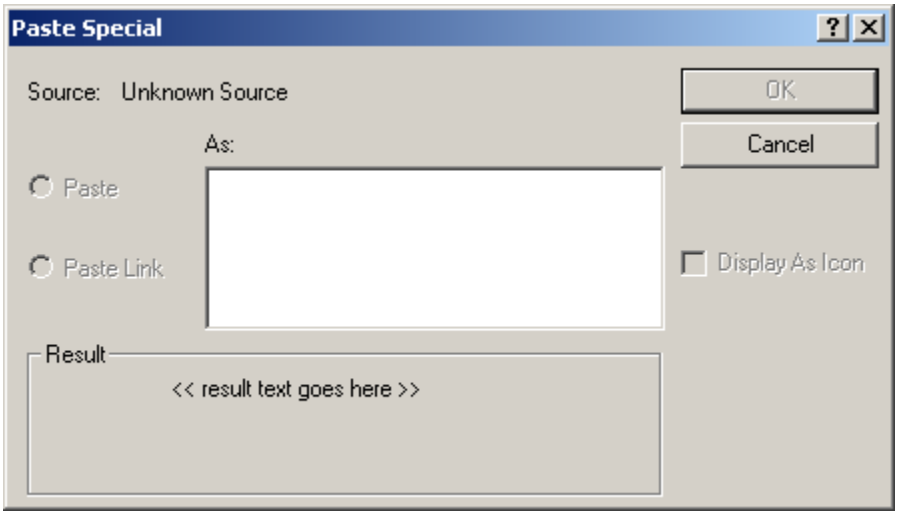
Keys: **SHIFT+INS**

Paste Special

Select **Paste Special** from the **Edit** menu to paste objects in a specified format when more than one format is available. For example, some applications may copy images as both images and metafiles. The **Paste Special** dialog box, shown below, allows you to choose whether to paste the object as an image or metafile or as a device independent image.

If the **Display As Icon** box is checked, the image is displayed as an icon on your desktop.

You can paste the object if **Paste** is selected, or you can paste the link to the object if **Paste Link** is selected.



Paste Special Dialog Box

Duplicate



Select **Duplicate** from the **Edit** menu to duplicate a selected object.

To duplicate an object:

1. Select the object. A box with square handles surrounds the object.
2. Select **Duplicate** from the **Edit** menu, or click the **Duplicate** button on the **Main** toolbar. When an object is duplicated, all of its attending properties accompany the duplicated object.

Shortcuts

Toolbar: 

Keys: **CTRL+D**

Format Painter



Select **Format Painter** from the **Edit** menu to copy the visible formatting of one object to another object.

To format an object:

1. Select the object(s). A box with square handles surrounds the object.
2. Select **Format Painter** from the **Edit** menu, or click the **Format Painter** button on the **Main** toolbar. Then click on a second object or more objects) in the display. The formatting of the initially selected object is applied to the second object.

Shortcuts

Toolbar: 

Select All

Select All Objects

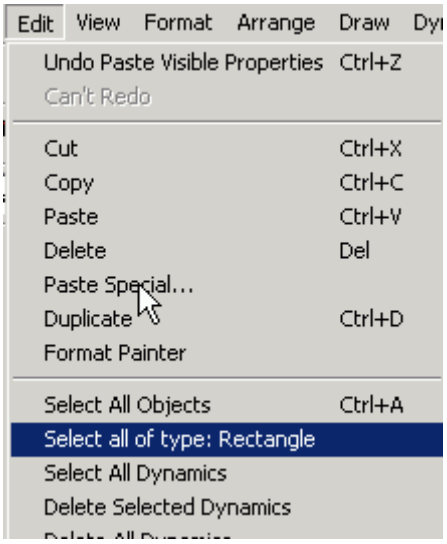
Choosing **Select All Objects** from the **Edit** menu highlights all objects in the display.

Select All of Type

Choosing **Select All of Type** from the **Edit** menu allows you to select all objects of a specific type contained inside the display. The type selected depends on the tool currently selected on the toolbar and can be one of the following:

- Line
- Rectangle
- Ellipse
- Arc
- Poly-line
- Text
- Picture
- Metafile
- Process Point
- Time/Date
- Push Button
- Check button
- Radio Button

For example, if the **Rectangle** button is currently selected on the **Draw** toolbar, then the **Select All of Type** item on the **Edit** menu appears as shown in the figure below.



'Select All of Type' Command on Edit Menu

Select All Dynamics

Choosing **Select All Objects** from the **Edit** menu highlights all dynamic objects in the display (e.g. process points, time/date, etc.).

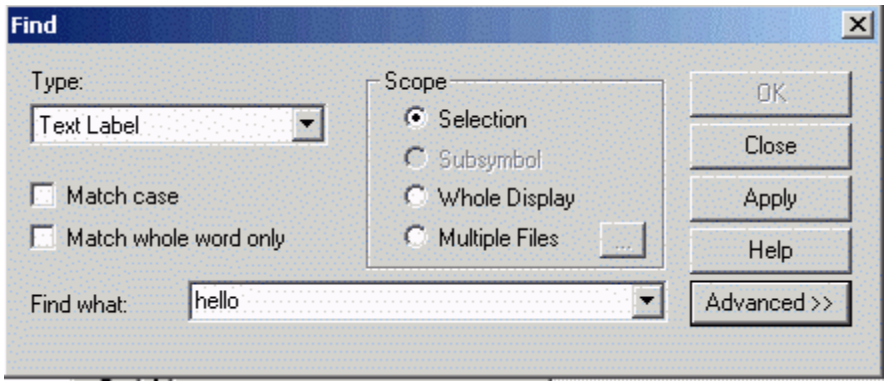
Find, Report, and Replace

The **Find** and **Report** functions on the **Edit** menu allow you to find and report strings in GraphWorX32.

Selecting **Find** from the **Edit** menu opens the **Find** dialog box, as shown in the figure below. Enter a string to search for in the **Find What** field.

If **Match Case** is checked, this function finds/replaces only text strings that match the case of the characters. Otherwise the command finds/replaces strings with either uppercase or lowercase character that match the character in the **Find What** string.

If **Match whole word only** is checked, this function finds/replaces the entire text strings. Otherwise the command find/replace selects any string that contains a **Find What** substring.



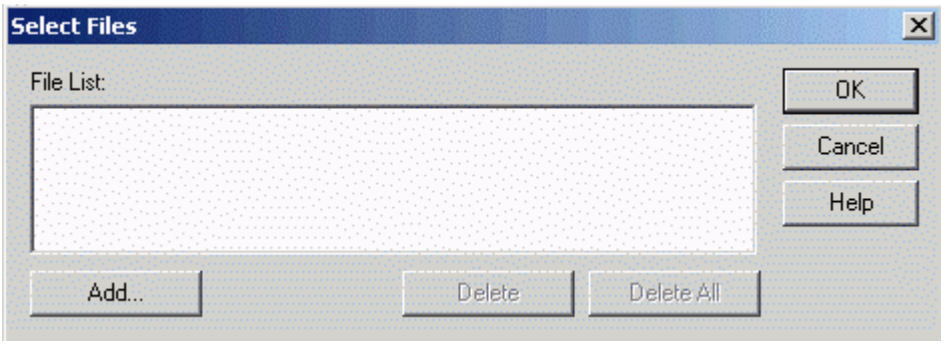
Find Dialog Box

Choose one of the following object types to search from the drop-down list under **Type:**

- Dynamic Tag
- Text Label
- Object Name
- Share Keyword
- Description
- Custom Data
- File Name

Under **Scope,** you can limit the search to one of the following:

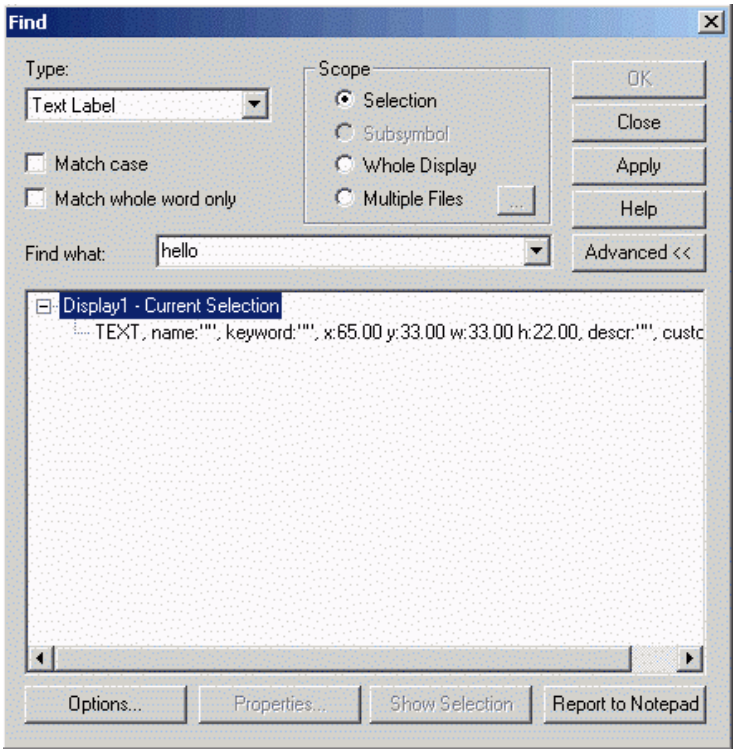
- **Selection:** Searches the object that is currently selected in the display.
- **Subsymbol:** Searches a subsymbol in the display.
- **Current Layer:** Searches the current layer (in displays with multiple layers).
- **Whole Display:** Searches all items throughout the entire display.
- **Multiple Files:** Searches a group of display files. To specify the files to search, click the **...** button next to the **Multiple Files** radio button. This opens the **Select Files** dialog box, as shown in the figure below. Click the **Add** button to choose a file. All selected files are shown under the **File List.**



Selecting Multiple Files to Search

Click the **Apply** button on the **Find** dialog to perform the search. To view the search results, click the **Advanced** button. This expands the **Find** dialog box to show the search results, as shown in the figure below. The results are displayed as a tree view. Selecting an item in the results tree and clicking the **Properties** button opens the Property Inspector for the selected item. Click the **Show Selection** button to view the item in the display. In the Multiple File scope, selecting an object not in the current display loads a new display with the object selected.

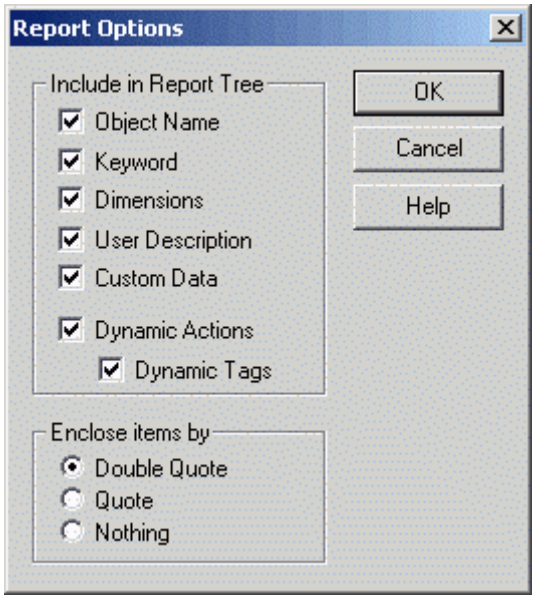
Note: If the type is **Object Name** and the **Find What** string is '*****', the results tree view displays all object hierarchy from current Scope (including the Multiple Files scope).



Find Dialog Box: Results View

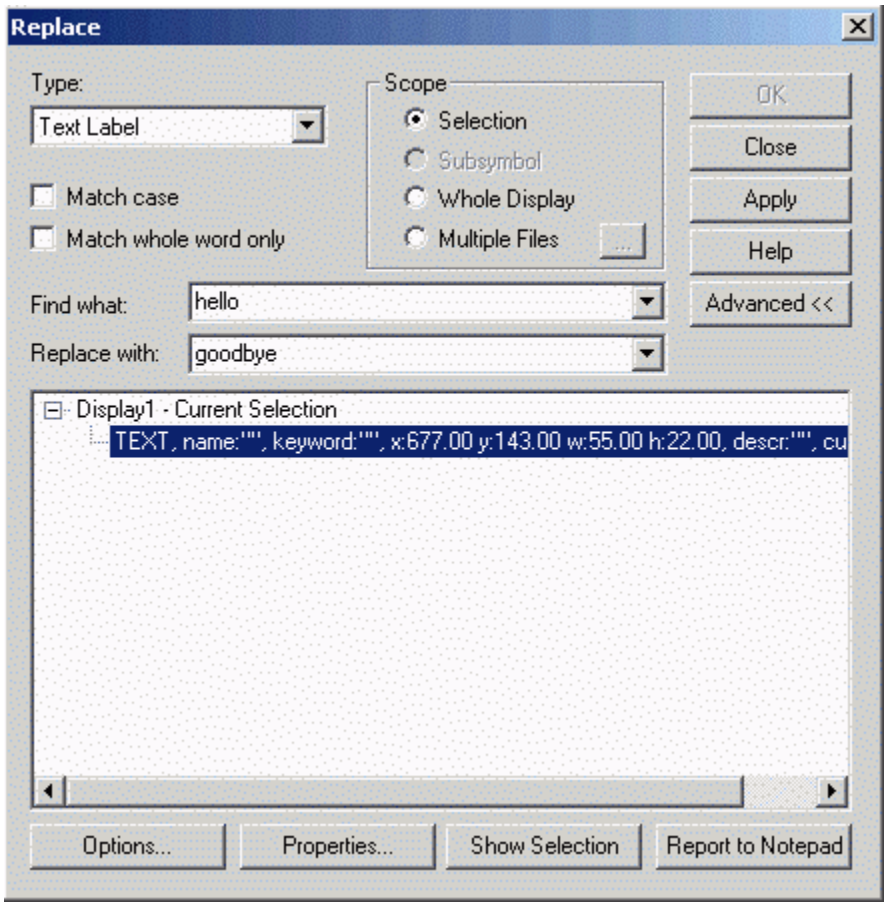
You can customize the results view by clicking the **Options** button. This opens the **Report Options** dialog box, as shown in the figure below. Choose the items to include in the results tree. You can also choose to identify items in the results by enclosing the items in double or single quotes.

To generate a report for the Find operation, click the **Report to Notepad** button on the **Find** or **Report** dialog boxes. This exports the search results to a text file.



Report Options Dialog Box

The **Replace** operation works in a similar way and replaces the **Find What** string(s) with the **Replace With** string(s). All dialog options are the same, except that the **Replace** dialog box has a field called **Replace With,** which also allows you to select from the **Tags Menu** when **Dynamic Tag** is selected under **Type.**



Replace Dialog Box

Find/Replace Dialog Box Parameters

Parameter	Description
Type	Specifies a type of string where the Find and Replace commands work. All the strings can be modified for every object or dynamic action through the Property Inspector dialog box.
Dynamic Tag	Finds/replaces tags (dynamics points).
Text Label	Finds/replaces labels of text objects.
Object Name	Finds/replaces names of objects/ dynamic actions.
Share Keyword	Finds/replaces global keywords used in shared objects.
Description	Finds/replaces based on a description.

Custom Data	Finds/replaces custom data.
File Name	Finds/replaces file names.
Scope	Specifies where to find/replace items.
Selection	Searches in the currently selected item in the display. If no object is selected, this option is disabled.
Subsymbol	Searches a subsymbol in the display.
Whole Display	Searches all items throughout the entire display.
Multiple Files	Searches a group of display files. To specify the files to search, click the **...** button next to the **Multiple Files** radio button.
Match Case	If checked, this function finds/replaces only text strings that match the case of the characters. Otherwise the command finds/replaces strings with either uppercase or lowercase character that match the character in the "Find What" string.
Match Whole Word Only	If checked, this function finds/replaces the entire text strings. Otherwise the command find/replace selects any string that contains a "Find What" substring.
Find What	Specifies the search text. A drop-down list allows you to select from all strings listed. This set of strings is determined by the Scope and Type fields. Wildcard*s* are allowed (see Wildcards below). Note that if the type is Object Name and the Find What string is '*', the results tree view displays all object hierarchy from current Scope (including multiple files scope).
Replace With	Specifies the string to replace the characters found. A drop-down list allows you to select from all possible strings. This set of strings is determined by the Scop*e* and by the Type radios. Wildcard*s* do not work here (they have no special meaning).
Results View	Tree view shows the result of the find/replace operation. All the object hierarchy that matches the Scope and "Find What" string is displayed as a tree control. If a leaf (an object or symbol) is selected, clicking the **Show Selection** button selects the specified object in the display. Note that in the multiple file scope, selecting an object not in the current display loads a new display with the object selected.
OK	Starts the find/replace operation. The matching objects are selected in the display and the dialog is closed.

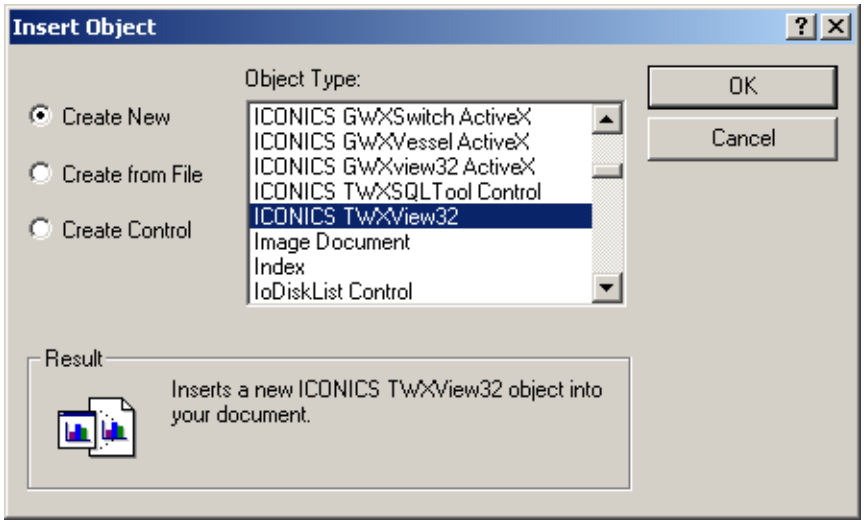
Cancel/Close	Cancels the dialog and closes it. If the Apply button is clicked, the label is changed from Cancel to Close, and the state of the dialog from the last Apply is saved.
Apply	Starts the find/replace operation. The dialog is not closed so the result can be inspected in the results tree view.
Advanced	Toggles the results tree view and enables the Options, Properties, Show Selection, and Report to Notepad buttons.
Options…	Opens the **Report Options** dialog box, which customizes the results view settings.
Properties	Opens the Property Inspector for the selected item in the results view.
Show Selection	Displays the selected item in the results view.
Report to Notepad	Exports the search results to a text file.

Wildcards

"Wildcard characters" '*' and '?' can be used in a "Find What" string. Character '*' can be used to match a group of characters. Character '?' can be used instead of one character. Display all objects from current scope by choosing the Type - Object Name and entering an asterisk (*) in the "Find What" combo.

Insert Object

Selecting **Insert New Object** from the **Edit** menu opens the **Insert Object** dialog box, shown below. This function inserts and embeds an OLE object, such as a chart or an equation, into the current display. The application in which the object was created becomes active on the screen. The **Insert Object** command also allows you to insert an ActiveX control into the current display. The property sheet for that ActiveX control is automatically opened after inserting the ActiveX control.

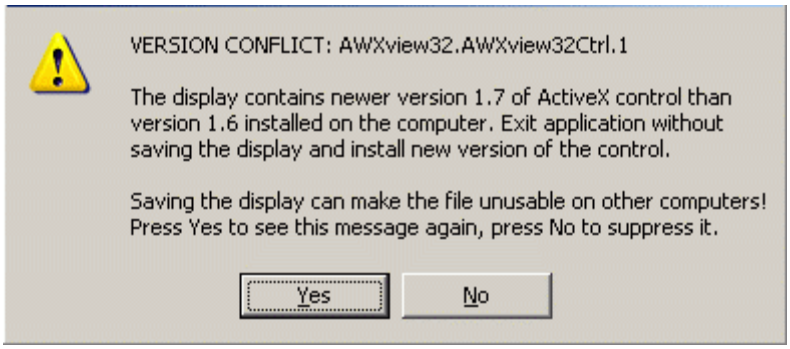


Insert Object Dialog Box

To insert a new object into the display:

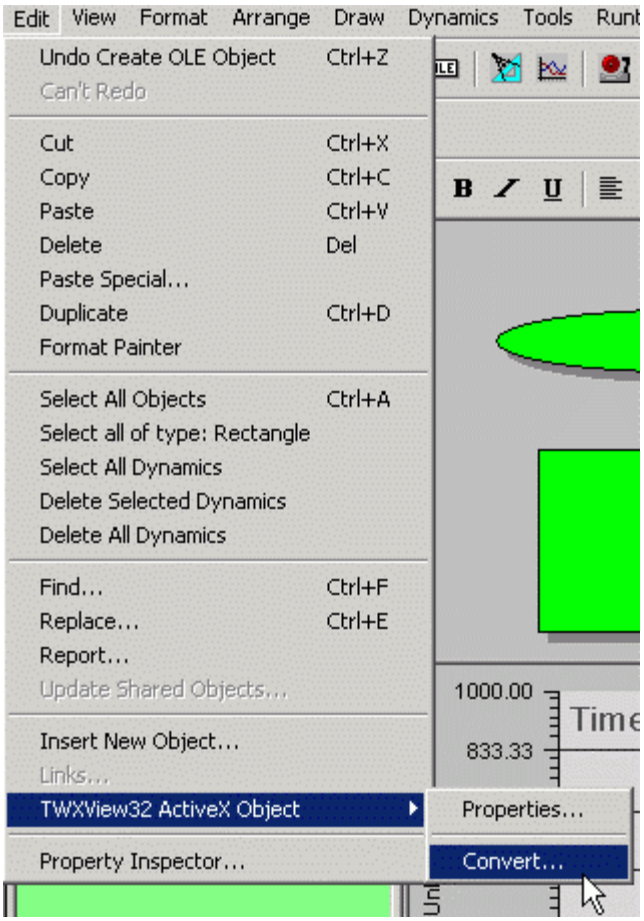
1. Choose either **Create New** or **Create From File. Create New** inserts a new OLE object from the list provided. Some of these objects have default values, which can be changed using a property editor. When **Create From File** is selected, you can enter the file name in the field provided, or you can click the **Browse** button to search for the file. When **Create Control** is selected, you can click the **Add Control** button to browse for an object.
2. Select the object type.
3. Click **OK.**

GraphWorX32 checks the version of ActiveX controls embedded in the display and installed on the computer and reports any potential conflicts, as shown in the figure below.



Version Conflict With Embedded ActiveX Control

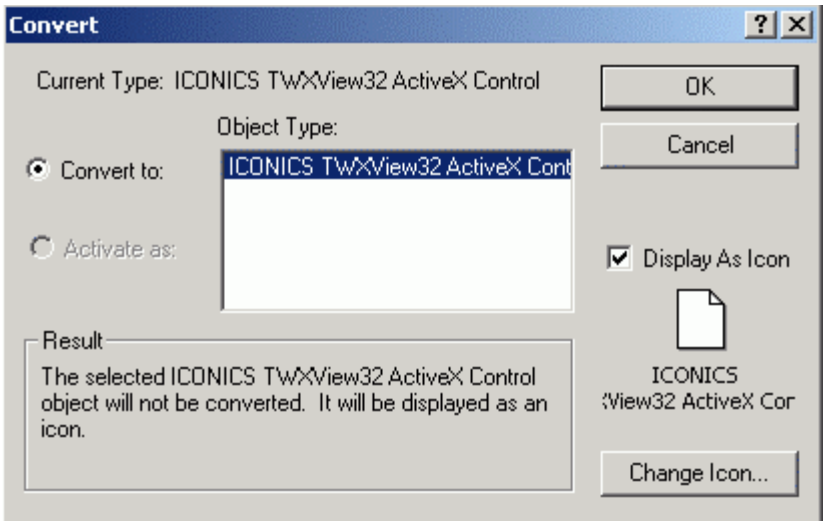
Once an object (e.g. TrendWorX32 Viewer ActiveX) is inserted into the display, the Object properties are enabled on the **Edit** menu, as shown in the figure below. Selecting **Object > Properties** from the **Edit** menu opens the Properties dialog box for the object.



Object Commands on Edit Menu

Selecting **Object > Convert** from the **Edit** menu opens the **Convert** dialog box, as shown in the figure below. To convert an object, select the **Object Type** and choose **Activate as.**

If an object is large and you want to reduce the amount of space that the object takes up in the display, check the **Display As Icon** check box. This replaces the object in the display with an icon of your choice. To choose an icon different from the standard icon, click the **Change Icon** button.



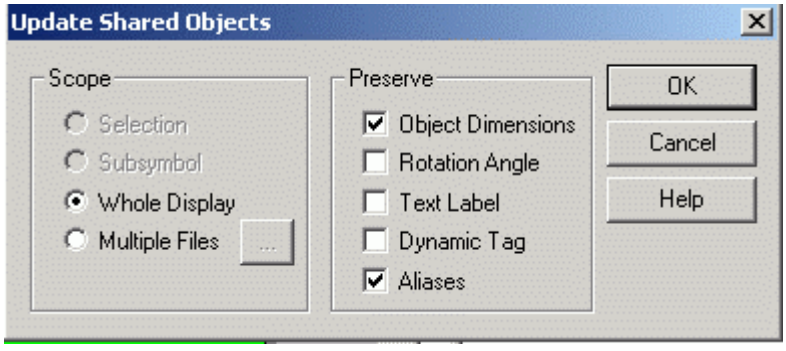
Converting an Object

Update Shared Objects

The **Update Shared Objects** function is accessible from the **Edit** menu. Selections made in the Property Inspector determine shared objects. An object can be shared by using the **Share Keyword** feature in the Property Inspector. It is made to belong to the desired group by entering a keyword shared by all the objects of one group. Any change can be made to an object of the group and can be applied to the rest of the objects in the specified group using **Update Shared Objects.**

A least one object from a group must be changed to enable the **Update Shared Objects** command. Thus, if several objects are integrated into a symbol, that symbol is duplicated. If the properties of one object in the symbol group are changed, the changes are applied globally to the duplicated object by using the **Update Shared Objects** function, once a share keyword has been specified.

Selecting **Update Shared Objects** from the **Edit** menu opens the **Update Shared Objects** dialog box, as shown in the figure below.



Update Shared Objects Dialog Box

Update Shared Objects Dialog Box Parameters

Parameter	Description
Scope	Determines the operating space for the operation.
Selection	Updates the currently selected item in the display.
Subsymbol	Updates a subsymbol (see menu item Edit Symbol and Subsymbol Editing capability); if there is no subsymbol edited, this option is disabled.
Whole Display	Updates the entire display; any subsymbol editing is left and replaced by a root edit level.
Multiple Files	Updates a group of file display files. To specify the files to search, click the **...** button next to the **Multiple Files** radio button. Any subsymbol editing is left and replaced by root level. If there are changes in the current document, a dialog appears asking to save them or cancel the operation.
Preserve	If you do not want to update all aspects of an object, you can preserve (not update) any of the following by checking the appropriate check boxes: Object Dimensions, Rotation Angles, Text Labels, Dynamic Tags, and Aliases.

Note: You can preserve or replace the data source and the alias resolutions independently. When the **Dynamic Tag** check box is checked, the **Aliases** check box is irrelevant and therefore disabled because the data bindings will not be affected. If the **Dynamic Tag** check box is not checked then, depending on the aliases state, the alias resolutions are preserved (**Aliases** check box checked) or replaced (**Aliases** check box not checked).

For example:

1. Create a rectangle.
2. Assign a keyword **RECT** to the **Share Keyword** field in the **Property Inspector.**
3. Attach a Size action and set the data source to <<tag>>.
4. Duplicate the rectangle (CTRL+D).
5. Resolve the alias <<tag>> in the first rectangle to **ICONICS.Simulator.1\SimulatePLC.Sine.**
6. Resolve the alias <<tag>> in the second rectangle to **ICONICS.Simulator.1\SimulatePLC.Ramp.**

The task now is to modify the data source in these shared rectangles from <<tag>> to \\<<machine>>\<<tag>>, while we want to keep the existing alias resolutions to sine and ramp.

The solution is to update the data source in one of the rectangles and then, while having this rectangle still selected, update other shared objects by selecting **Update Shared Objects** from the **Edit** menu.

After replacement both objects should contain the updated data source \\<<machine>>\<<tag>>, but they should keep the alias resolution for the <<tag>> alias – to sine and ramp.

Note if there was an alias resolution assigned to the <<machine>> before updating other shared objects, other shared objects will be updated as expected.

Where and How to Use Shared Objects

Consider a display with many inputs and outputs. If all input label objects are marked as shared with, for example, INPUT keyword, and all output label objects are marked as shared with, for example, OUTPUT keyword, it is very fast and easy to modify the color, size, 3-D effect, and shadow for all of them. Just do the change on one object and apply it to the rest. (Remember to check the **Preserve Text Labels** check box.)

Consider a project with many pumps. If the displays in the project are created carefully from the beginning, and all desired pumps are marked as shared with same keyword, it is fast and easy to change the appearance. If the pumps are connected to different signals, remember to check the **Preserve Text Labels** check box on the **Update Shared Objects** dialog box. For changes to tags, consider using the Find/Replace capability with wildcards to select the desired group of signals to be searched or replaced.

Consider a company logo or some similar pictures placed in many displays in different positions (if placed in the same position, templates should be the right tool for that job). By one operation, all of them can be modified or increased in size. To make only one larger, use the **Update Share Objects** function, and remember to uncheck **Preserve Object Dimensions** on the **Update Shared Objects** dialog box.

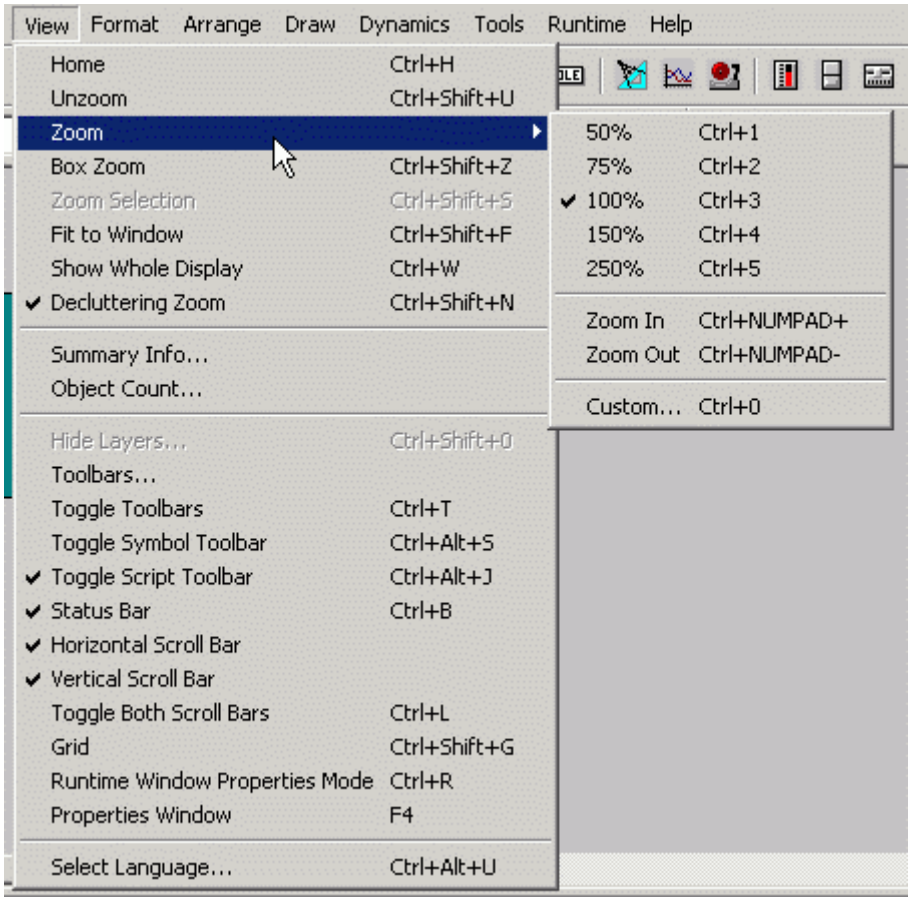
Chapter 6: Viewing Files

Contents

View Functions
Home View
Zoom Functions
Summary Information
Viewing Object Count Statistics
Hide Layers
Toolbars
Viewing Current Coordinates on the Status Bar
Scrolling
Grid
Runtime Window Properties Mode
Properties Window
Select Language

View Functions

The following sections describe the **View** menu functions shown below. Many functions can also be accessed through the **View** toolbar.



View Menu



View Toolbar

Home View



The **Home** view is defined as the view in which you last saved the display.

To restore the viewing area to the home view, select **Home** from the **View** menu. The display work area that in which the display was last saved is restored to the screen. The zoom factor appears on the bottom right of the screen.

Zoom Functions

The **Zoom** factor is indicated in the bottom right portion of the status bar. 100% zoom, 1 pixel = 1 unit is the default.

Zoom
Custom Zoom
Unzoom
Box Zoom
Zoom Selection
Fit to Window
Show Whole Display
Decluttering Zoom
Mouse Zoom

Zoom

To zoom a display, select **Zoom** from the **View** menu, or select one of the buttons shown below from the **View** toolbar. The **Zoom** submenu on the **View** menu allows you to choose from a default zoom value, to zoom in or out, or to type in a custom zoom percentage.

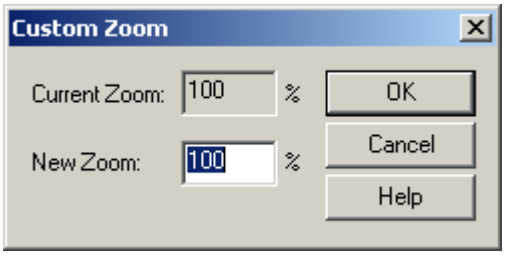

Zooms in by increments of 50 percent.


Zooms out by increments of 50 percent.


Specifies a custom percentage to zoom.

Custom Zoom

To set a custom zoom percentage, select **Zoom > Custom** from the **View** menu. This opens the Custom Zoom dialog box. Enter a percentage in the **New Zoom** field, and then click **OK.**



Setting a Custom Zoom Percentage

Unzoom



The unzoom function acts as an undo for zoom. Select **Unzoom** from the **View** menu, or click the **Unzoom** button on the **View** toolbar.

Box Zoom



The **Box Zoom** command allows you to select an area on which to zoom.

To use the box zoom function:

1. Select **Box Zoom** from the **View** menu. The cursor appears as a magnifying glass, indicating the zoom option is enabled.
2. Click the left mouse button and drag to draw a box around the area in the display selected to zoom, and the release the mouse button. Or, click the left mouse button without dragging to zoom in (plus 50 percent) on the point on which you clicked. Click the right mouse button without dragging to zoom out (minus 50 percent) on the point clicked on.
3. The area appears in zoomed view. You can zoom in numerous times.
4. Select **Home** from the **View** menu when you are done viewing that display area in the zoom mode and want to return to the home view. Or select **Unzoom** from the **View** menu to undo your zoomed views.

Setting Zoom Limits for Box Zoom

The Box Zoom feature allows you to select an area to which the current view should zoom. However, it also allows you to zoom in by left-clicking, or to zoom out by right-clicking. You can specify limits for the Box Zoom, so the operator using the Box Zoom does not run out of preferred zoom, using the following registry entries:

[HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\GWX32\Runtime Settings]

BoxZoomMaxPercent: DWORD .. maximal allowed zoom, by default 999 (%)

BoxZoomMinPercent: DWORD .. minimal allowed zoom, by default 1 (%)

Example:

To configure the Box Zoom to zoom between 30% and 150%, set BoxZoomMinPercent to 30 and BoxZoomMaxPercent to 150%.

Zoom Selection



Select **Zoom Selection** from the **View** menu, or click the **Zoom Selection** button on the **View** toolbar, to zoom in on the selected object(s).

Fit to Window



Select **Fit to Window** from the **View** menu, or click the **Fit to Window** button on the **View** toolbar, to size the display such that all objects in the display fit inside the window.

Show Whole Display



Select **Show Whole Display** from the **View** menu, or click the **Show Whole Display** button on the **View** toolbar to see the entire work area.

Decluttering Zoom

The **Decluttering Zoom** option on the **View** menu allows layers to be hidden when they are zoomed in or zoomed out beyond their zoom threshold. For this feature to work, you must set the zoom thresholds by selecting **Layers > Edit Layer Properties** from the **Format** menu and specifying the **Zoom Visibility Range** in the **Edit Layer Properties** dialog box.

Mouse Zoom

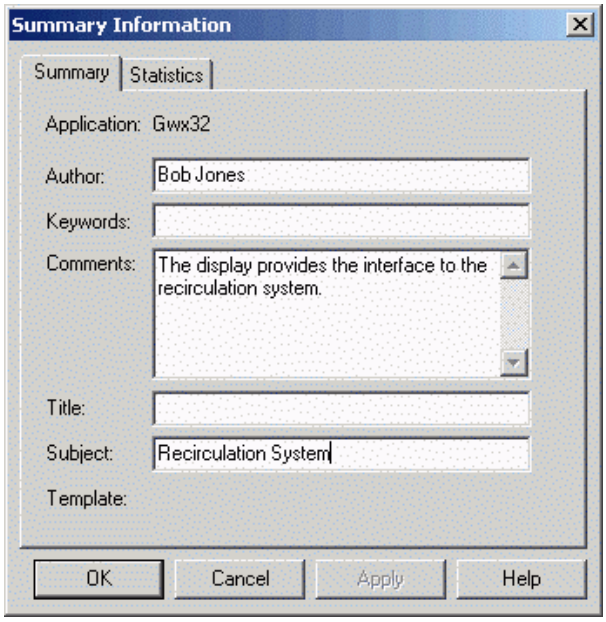
A mouse zoom feature has been added to GraphWorX32 to simplify the display navigation:

- To zoom in, press the **ALT** key and roll the mouse wheel forward.
- To zoom out, press the **ALT** key and roll the mouse wheel backward.
- To pan the screen press the **ALT** key, push the left mouse button, and drag the mouse.

The zoom is selection-sensitive so that if an object is selected, the display automatically recenters the screen to the object. This also applies to multi-selection of display objects.

Summary Information

Selecting **Summary Information** from the **View** menu opens the **Summary Information** dialog box, shown below.



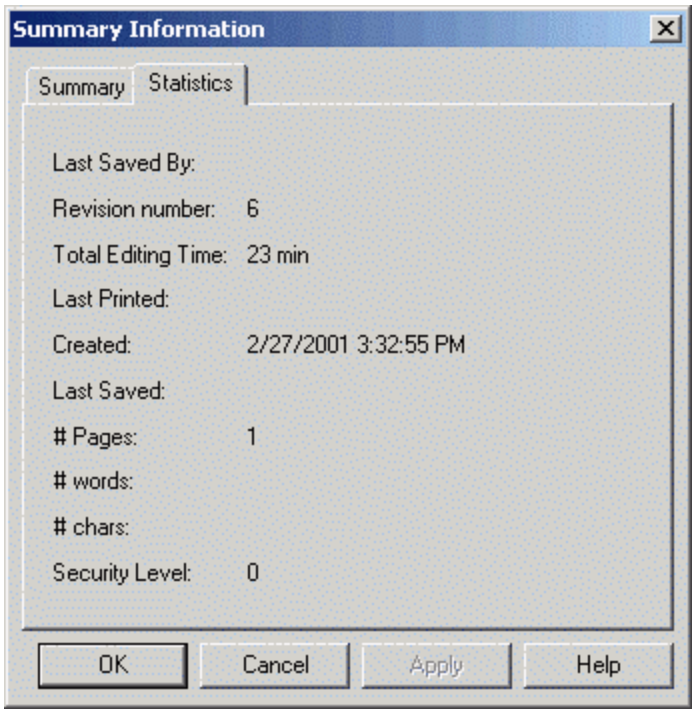
Summary Information Dialog Box

Summary Tab

In the **Summary** tab, you can fill in the details such as **Author, Keywords, Comments, Title,** and **Subject** in the spaces provided.

Statistics Tab

The **Statistics** tab, shown below, provides general information about the file currently on display.



Statistics Tab

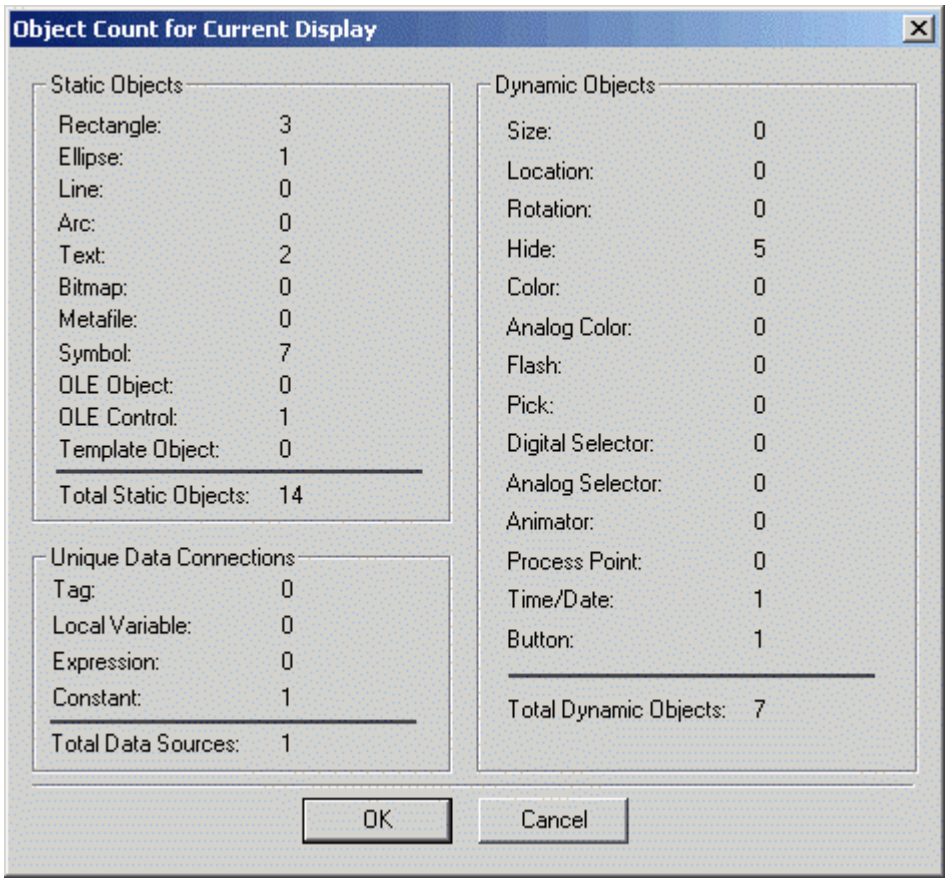
Note: Summary Information for a display can also be viewed in Windows Explorer. In Windows Explorer, right-click the display file and choose **Properties** from the pop-up menu. The **Summary** and **Statistics** tabs will appear in the dialog displayed by Windows Explorer.

Viewing Object Count Statistics

The **Object Count** function on the **View** menu displays statistics for objects, actions, and display sizes in your displays.

To display object statistics:

1. Select **Object Count** from the **View** menu. This opens the **Object Count for Current Display** dialog box, shown below, which contains data about static and dynamic objects.
2. Click **OK** to return to the display.



Viewing Object Count Statistics

Object Count Parameters

Parameter	Description
Static Objects	
Rectangle	Number of rectangles in the display.
Ellipse	Number of ellipses in the display.
Line	Number of lines in the display.
Arc	Number of arcs in the display.
Text	Number of text objects in the display.

Bitmap	Number of images in the display.
Metafile	Number of metafiles used in the display.
Symbol	Number of symbols used in the display.
OLE Objects	Number of OLE Objects in the display.
OLE Control	Number of OLE Controls in the display.
Template Object	Number of Template Objects in the display.
Total Static Objects	Total number of static objects in the display.

Unique Data Connections

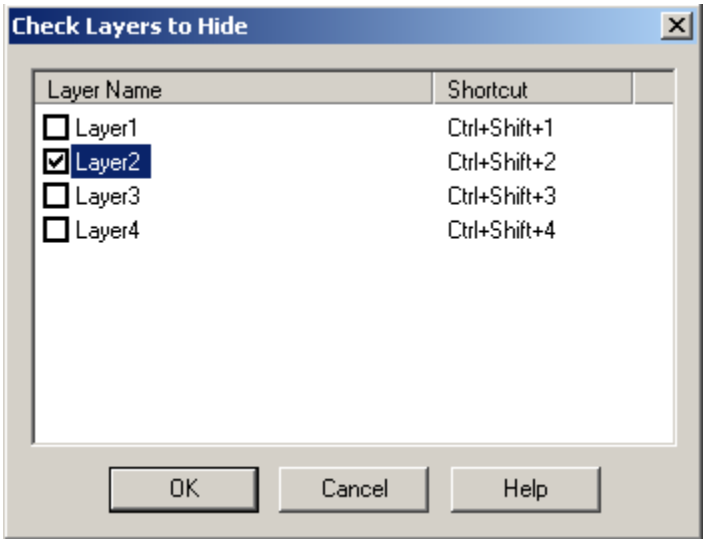
Tags	Number of Tags in the display.
Local Variable	Number of Local Variables in the display.
Expression	Number of Expressions in the display.
Constant	Number of Constants in the display.
Total Data Sources	Total number of data sources in the display.

Dynamic Objects

Size	Number of Size connections in the display.
Location	Number of Location connections in the display
Rotation	Number of Rotation connections in the display.
Hide	Number of Hide connections in the display.
Color	Number of Color connections in the display
Analog Color	Number of Analog Color connections in the display.
Flash	Number of Flash connections in the display.
Pick	Number of Pick connections in the display.
Digital Selector	Number of Digital Selector connections in the display.
Analog Selector	Number of Analog Selector connections in the display.
Animator	Number of Animator Actions connections in the display.
Process Point.	Number of Process Points in the display..
Time/Date	Number of Time and Date fields in the display.
Button	Number of pushbuttons in the display.
Total Dynamic Objects	Total number of dynamic objects in the display.

Hide Layers

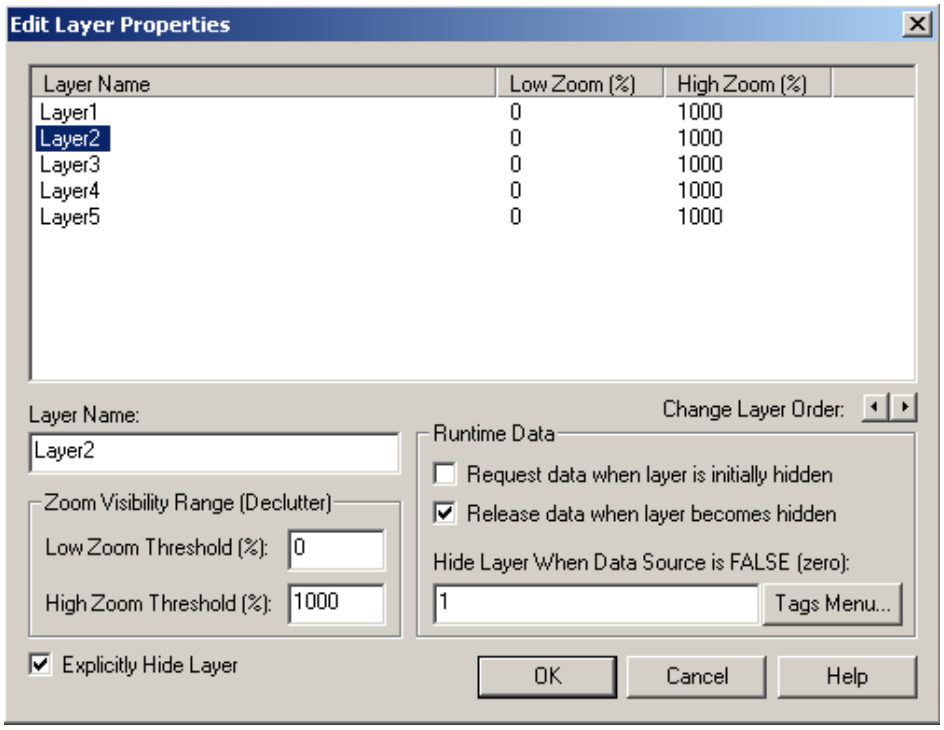
You can hide or show layers by selecting **Hide Layers** from the **View** menu or the keyboard shortcut combination **CTRL+SHIFT+0**. Both of these options open the **Check Layers to Hide** dialog box, shown below. Select a layer or layers to hide, and then click **OK.**



Selecting Which Layers to Hide

Layers may also be hidden due to the value of the tag to which they are attached, as well as if you select **Layers > Edit Layer Properties** from the **Format** menu and select the **Explicitly Hide Layer** check box in the **Edit Layer Properties** dialog box, as shown in the figure below. The **Explicitly Hide Layer** feature forces the layer to remain hidden regardless of whether the layer would be visible based on the zoom threshold. The layer will remain hidden until this option is turned off, at which point the layer may remain hidden depending on the attached tag value and zoom threshold. Some layers can hide appear based on other conditions, but an explicitly hidden layer is just hidden and never appears. If the layer is not explicitly hidden, then it may be hidden depending on the value of the attached tag and the zoom threshold (decluttering zoom).

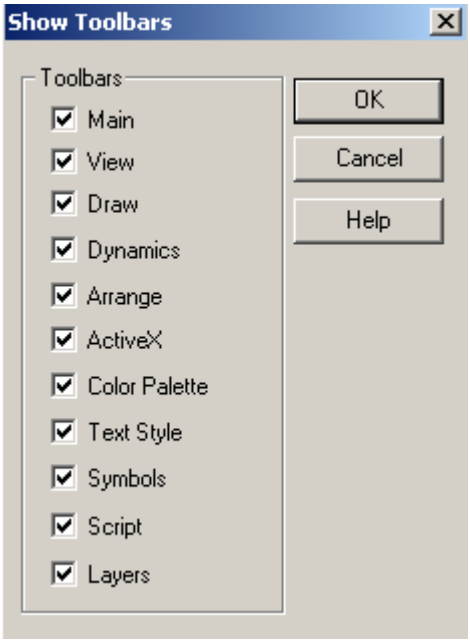
Each layer that you added during configuration is shown in the dialog box, along with the system generated shortcut key. Note that there is a check box to the left of each layer name. If this box is checked, that layer will be hidden during runtime mode.



Explicitly Hide Layers Function in Layer Properties

Toolbars

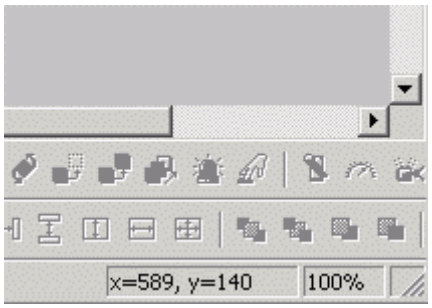
The **Toolbars** command on the **View** menu opens the **Show Toolbars** dialog box, shown below, which allows you to select which toolbars you want to be visible. The **Toggle Toolbars** command on the **View** menu hides or shows all of the toolbars at once.



Show Toolbars Dialog

Viewing Current Coordinates on the Status Bar

The current mouse coordinates are shown in the right-hand corner of the status bar. The current coordinates identify the position of the mouse pointer in the work area through X- and Y-coordinates, as shown in the figure below. This is helpful if you are working in pixels and need to define the mouse pointer position more accurately.



Viewing Mouse Coordinates on the Status Bar

Scrolling

The scrolling function allows you to scroll up and down your display using the scroll bar displayed on the right of your screen. It also allows you to scroll left and right using the horizontal scroll bar displayed at the bottom of your screen.

Scroll Bars

The **Horizontal Scroll Bar** and **Vertical Scroll Bar** commands on the **View** menu set the visibility of the horizontal and vertical scroll bars, respectively. The **Toggle Scroll Bar** command on the **View** menu shows or hides both scroll bars at once.

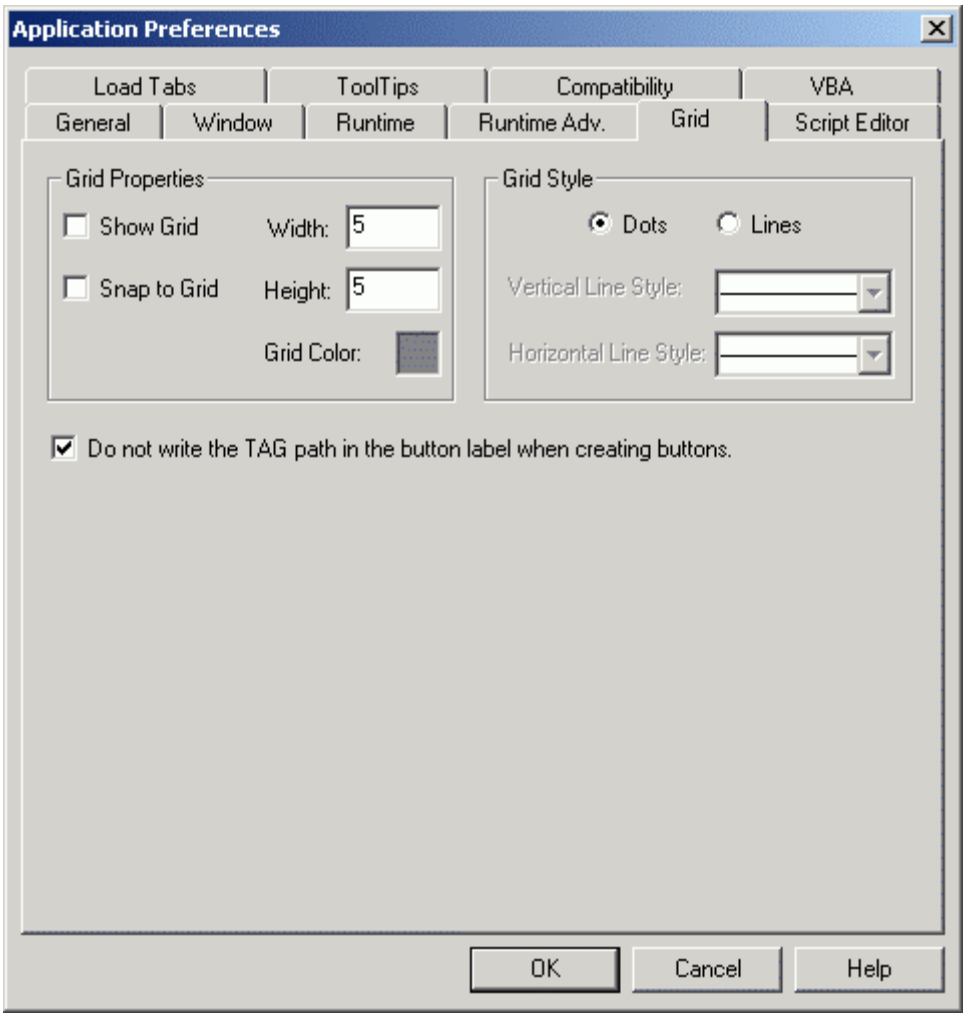
Grid

The **Grid** command on the **View** menu shows or hides the grid. The **Grid** tab in the **Application Preferences** and **Display Properties** dialog boxes, shown below, gives you flexibility with line style, color, and spacing.

The **Grid Size** is in arbitrary base units (depending upon the current zoom factor). When zoom is at 100%, the spacing is in pixels. The **Grid Size** specifies the grid spacing in terms of width and height.

The **Grid Style** allows you to choose a style of dots or dashes (lines). You can choose a **Vertical Line Style** or a **Horizontal Line Style** from the options available.

You can choose a **Grid Color** for the lines and dots of the grid. By checking the appropriate boxes, you can choose to **Show Grid** and **Snap to Grid.**



Application Preferences: Grid Tab

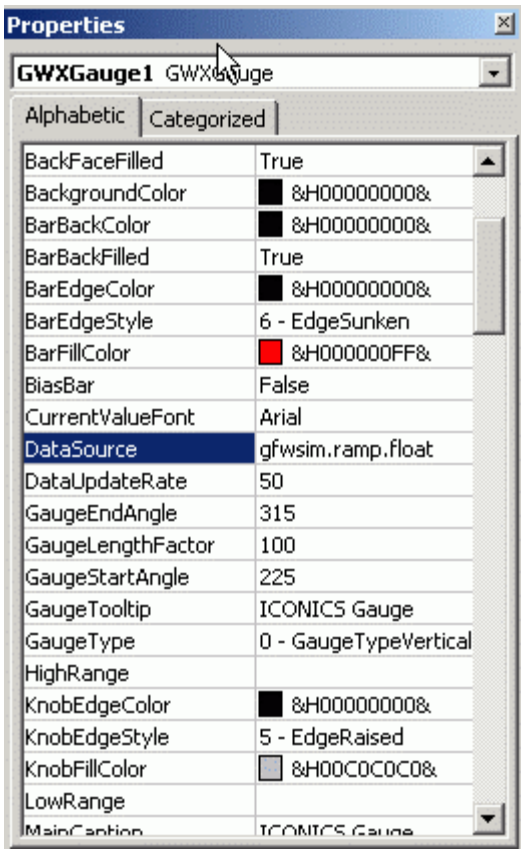
Runtime Window Properties Mode

Selecting **Runtime Window Properties Mode** from the **View** menu makes it possible to see what a display will look like in runtime mode without actually entering runtime. This helps you set the proper display dimensions and zoom factors. When the action toolbars are hidden, the configuration functions can be found in the **Main** menu or in the right-click pop-up menu.

Note: Pressing **CTRL+M** toggles the display between configuration and runtime modes.

Properties Window

Selecting **Properties Window** from the **View** menu opens the **Properties** list for the display, as shown below. This lists the property settings for the selected form or control. A **property** is a characteristic of an object, such as size, caption, or color.

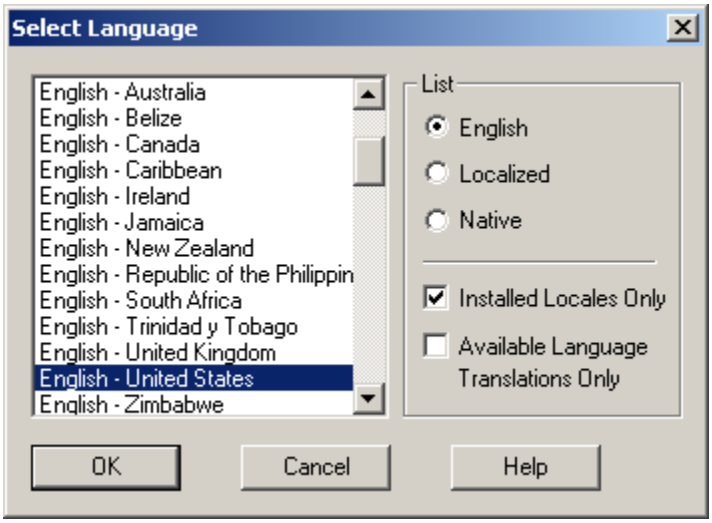


Properties Window

Select Language

The **Select Language** function on the **View** menu allows you to choose which language to use in your display. Choosing **Select Language** from the **View** menu opens the **Select Language** dialog box, shown in the figure below.

Note: A language resource .dll is required for language switching.



Select Language Dialog Box

Define the parameters listed in the table below. Then click **OK** to return to the work area.

Select Language Parameters

Parameter	Description
List	Lists available languages. Depending on which item you have selected, the view on the left will change. If **English** is checked, the languages will appear as their English name. If **Localized** is checked, the languages will appear with the native country in parentheses (for languages with several dialects only). When **Native** is checked, the languages are displayed the way they would be written in that language.
Installed Locales Only	If this is checked, local languages appear in the box.
Available Language Translations Only	Checking this box allows you to choose from available language translations only.

Chapter 7: Data Connections

Contents

Introduction to Data Connections
Tags
Expression Editor
Expressions
Local Variables
Constant Values
Aliasing Data Connections
Editing Data Source Connections

Introduction to Data Connections

This section discusses the interface of GraphWorX32 objects to the field Input/Output. The following topics will be discussed here:

- Making data connections with the Unified Data Browser
- Expressions
- Constants and local variables
- Creating generic reusable displays and symbols with aliasing

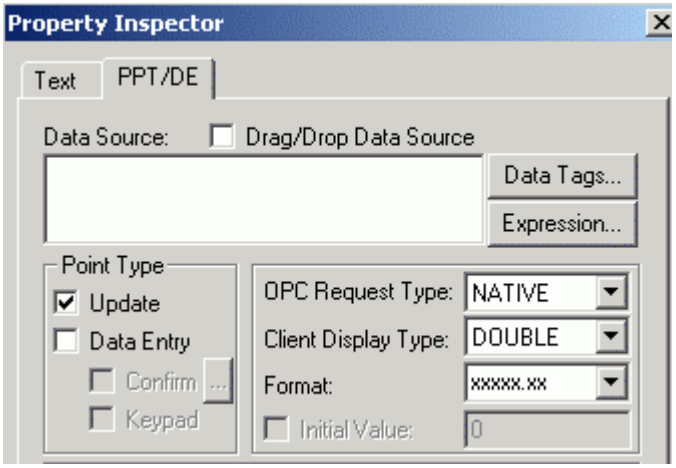
Dynamic animation is achieved by transforming visible objects based on specified data connections. Each dynamic object allows one or more data connections for its primary value(s). Some dynamics also allow range overrides; these range overrides are also data connections. A data connection is most likely to be a tag representing a value in the system. However, a data connection can also be a database connection, a constant value, a local GraphWorX32 variable, or a mathematical expression (which can include tags, constants, and local variables).

Data connections can be typed directly into the data source edit control in the configuration dialogs of the various dynamics. Text strings can also be dragged and dropped into this edit control from any drag source, which makes the drag-and-drop data available in the Windows text clipboard format. There are also two buttons:

Data Tags button - Opens the **Unified Data Browser.**

Expressions button - Opens the Expression Editor, which enables you to create and modify expressions.

Note: For more information about the Unified Data Browser, please see the Unified Data Browser help documentation. For more information about the Expression Editor, please see the Expression Editor help documentation.



Data Tags and Expression Buttons in Property Inspector

Tags

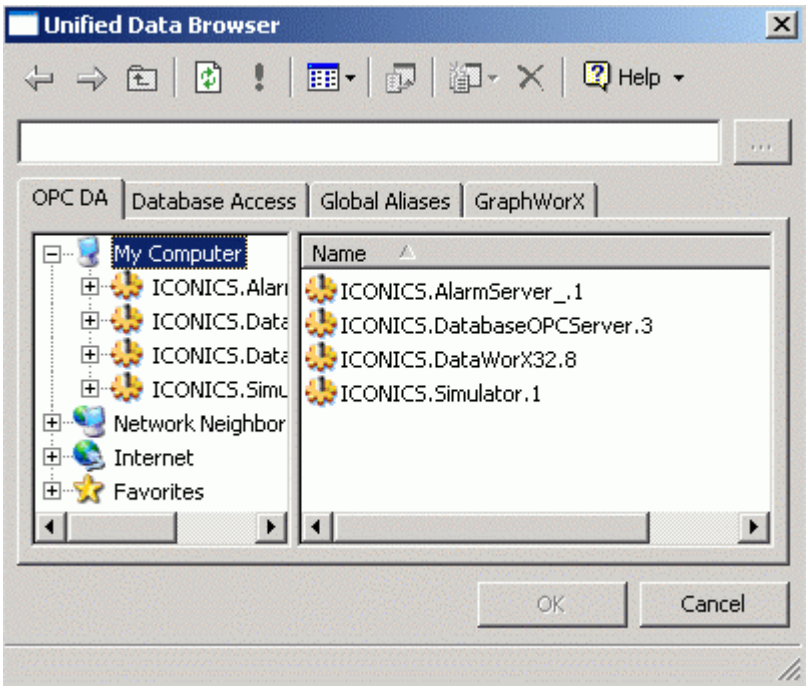
All nonnumeric data-connection strings that do not include the special tokens described later in this section are interpreted as tags. If your tag happens to be a string that is a number (a rare situation) and you do not want the tag to be auto-detected as a constant value, use the following syntax:

{{tag_name**}}**

Any tags that are not defined in the system at runtime will point fail.

Unified Browser

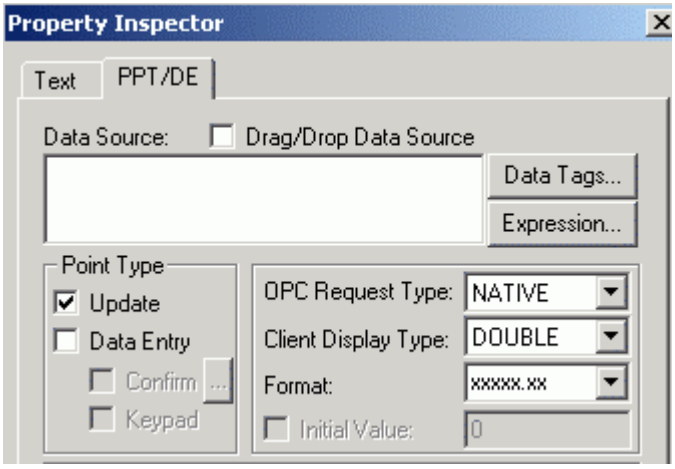
Clicking the **Data Tags** button in the Property Inspector opens the **Unfied Data Browser,** shown below. This feature allows you to select OPC tags and database connections. (See the Unified Data Browser documentation for more information.)



Unified Data Browser

Expression Editor

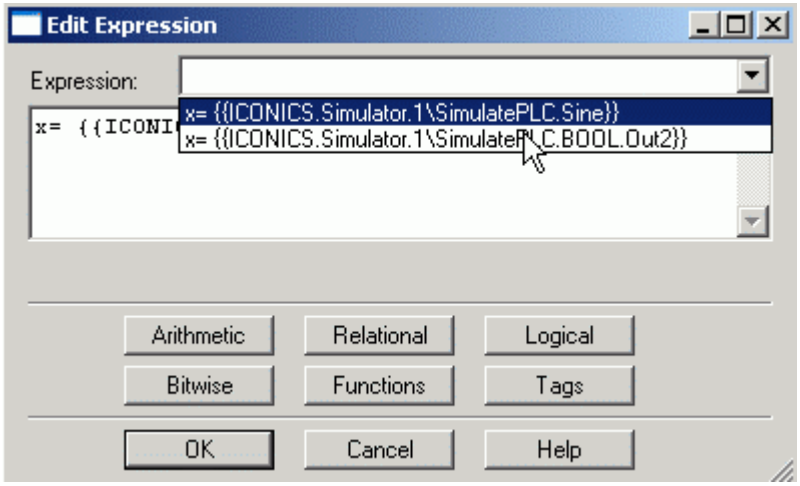
Clicking the **Expression** button in the Property Inspector opens the **Edit Expression** dialog box.



Expression Button in Property Inpector

The **Expression Editor,** shown below, is available to assist you in creating expressions for your GENESIS32 applications. The window is resizable and can be stretched as well as maximized or minimized. The drop-down list at the top of the **Edit Expression** dialog box keeps track of the last 50 expressions you have entered. The expression entered most recently is the first one in the drop-down list.

Note: For more information about the Expression Editor, please see the Expression Editor help documentation.



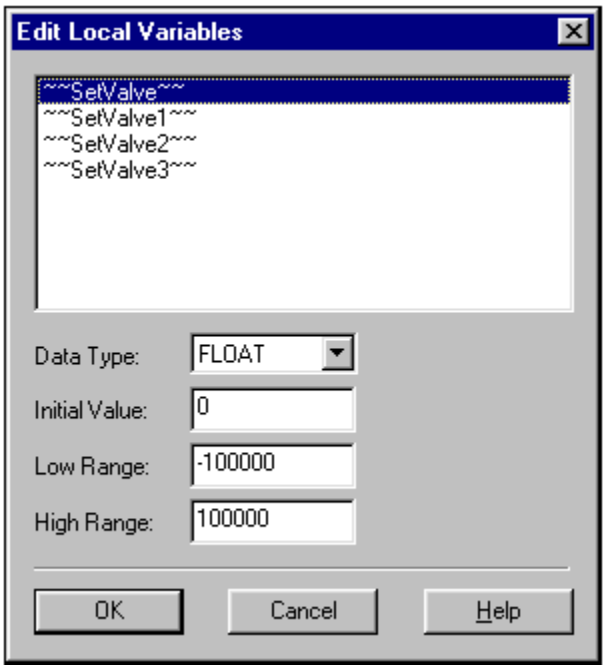
Edit Expression Dialog Box

Local Variables

To indicate that a data connection is a local variable, use the following syntax:

~~local_variable_name**~~**

Selecting **Edit Local Variables** from the **Dynamics** menu opens the **Edit Local Variables** dialog box, shown below. Here you can change the default settings for local variables.

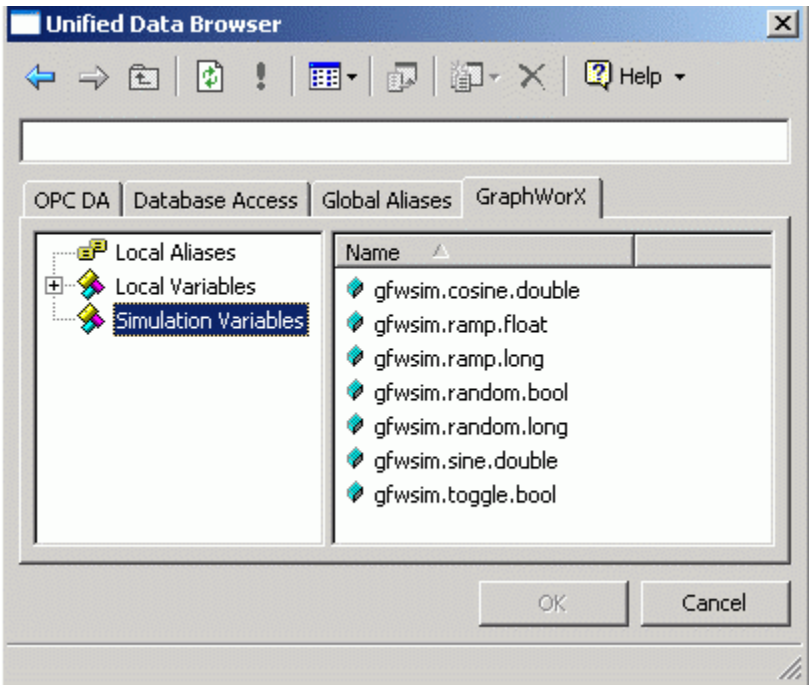


Edit Local Variables Dialog Box

Parameter	Description
Data Type	Specifies the data type of the local variable.
Initial Value	Sets the initial value of the local variable.
High Range and Low Range	Sets a range of values for the local variable.

The scope of local GraphWorX32 variables is confined to the display in which they are defined. Typically, local variables would be used as values for animation effects (i.e. simulated values that do not need to be defined at the system level).

GraphWorX32 includes several predefined local simulation variables. You can access a list of these variables by clicking on the **Data Tags** button in the Property Inspector. This opens the Unified Data Browser. Go to the the **GraphWorX** tab and select **Simulation Variables.** This lists all available simulation variables, as shown in the figure below. These simulation variables are useful for testing display animations when an OPC server is not available.



GraphWorX32 Simulation Variables

Constant Values

GraphWorX32 automatically recognizes if an entered data-connection string is a number and interprets that data connection as a constant value. If you want to enter a constant string value (a non-numerical constant), use the following syntax:

$"**constant_string_value**"$

This syntax prevents the string from being auto-detected as a tag name.

Aliasing Data Connections

Overview of Aliasing
Object-Level Aliasing
Edit Alias Table User Interface
Runtime Aliasing
Second-Level Aliasing
Global Aliasing
Language Aliasing

Overview of Aliasing

Aliasing is the ability to define a string (usually a short name) to represent another string (usually part or all of a tag name). You can access the **Edit Aliases** command on the **Dynamics** menu. You can also access it by right-clicking on an object in the display and selecting **Edit Aliases** from the pop-up menu.

When including an alias name in a data connection, the alias name should be surrounded by the special delimiters "**<<**" and "**>>**". These delimiters are used to identify the beginning and end of the alias name. For example:

[opcserver].<<tank>>.out.

In this case "tank" is the alias name.

Second-level aliasing allows an alias name to be entered as the alias definition. This second alias will be resolved prior to the set-aliases command.

Object-Level Aliasing

Objects that utilize data connections can maintain their own alias table. Each dynamic object can have an alias table. This alias table is automatically created when you enter a data source string containing alias names (the table is automatically populated with entries for each alias name used in the data connections of the dynamic object). By default, the alias replacement string is set equal to the alias name. If the alias name is equal to the alias replacement string, the alias will not be resolved. You are only allowed to change the replacement strings in object-level alias tables. The alias names in the table are controlled by the alias names used in the data-source strings.

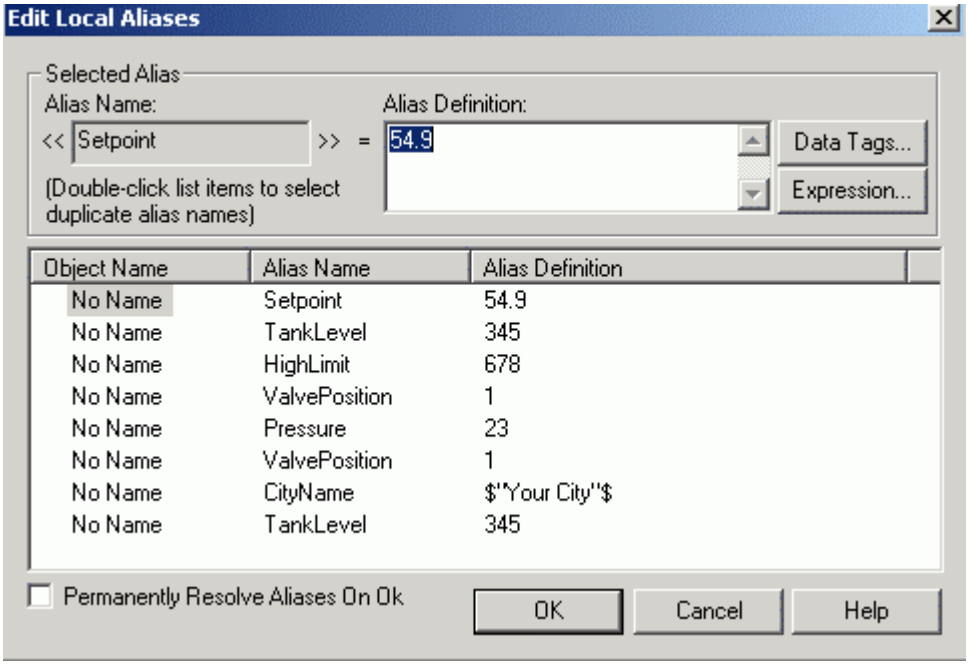
To edit object-level aliases in GraphWorX32, select one or more visible objects and then choose **Edit Aliases** from the **Dynamics** menu. This opens the **Edit Aliases** dialog box. All of the alias tables for all the dynamics attached to all of the selected objects are merged and displayed in the **Edit Aliases** dialog box. If the selected objects have object names defined, those names are used to identify which alias names belong to which objects. If no objects are selected when you choose to edit aliases, all aliases in the entire display appear in the **Edit Aliases** dialog box.

Editing Aliases

To open the **Edit Aliases** dialog box, shown below, select **Edit Aliases** from the **Dynamics** menu. This allows you to modify object-level alias tables and to change alias definitions for each alias name entry.

Multiple items in the list control can be selected simultaneously to allow multiple entries to be modified at once. Double-clicking on any alias name automatically selects all duplicate alias names in the list.

To sort the list of aliases by a certain category, simply click the column heading and the list will be arranged according to priorities in that column.



Edit Aliases Dialog Box

Edit Aliases Parameters

Parameter	Description
Alias Definition	Changes the definitions for all selected alias names.
Data Tags button	Launches the **Unified Data Browser.**
Expressions button	Launches the **Expression Editor.**
Permanently Resolve Aliases on OK check box	Aliases are normally resolved when entering runtime mode (the alias names are restored when you return to configuration mode). Checking this box resolves the aliases in configuration mode (consequently replacing the original aliases permanently).

Note: The alias replacement strings are modifiable during runtime mode via OLE Automation.

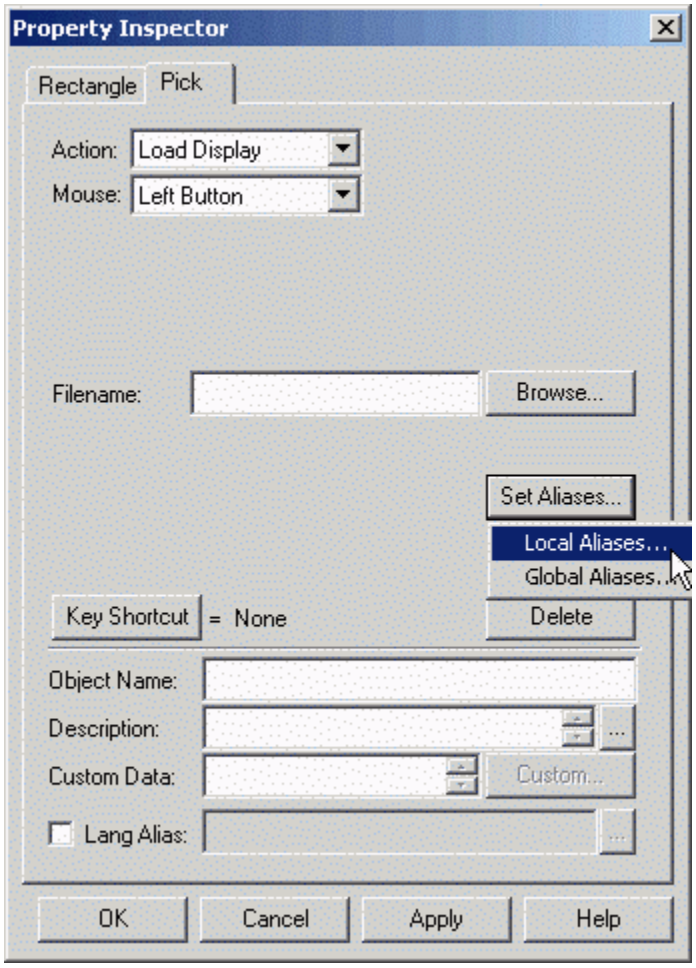
Runtime Aliasing

Runtime aliasing encompasses a variety of ways to change a display's aliases during runtime. You can specify a collection of aliases to be set as part of a given command. For example, when using a pick action to load a display, you can specify aliases to be set in the display that is about to be loaded. These aliases will be set before any tags in the new display are requested. The aliases to be set can be specified directly in GraphWorX32 or can come from an external tab-delimited text file (such files can be exported from Excel, created in Notepad, or created using GraphWorX32's Alias File Editor).

Commands with which aliases can be associated include:

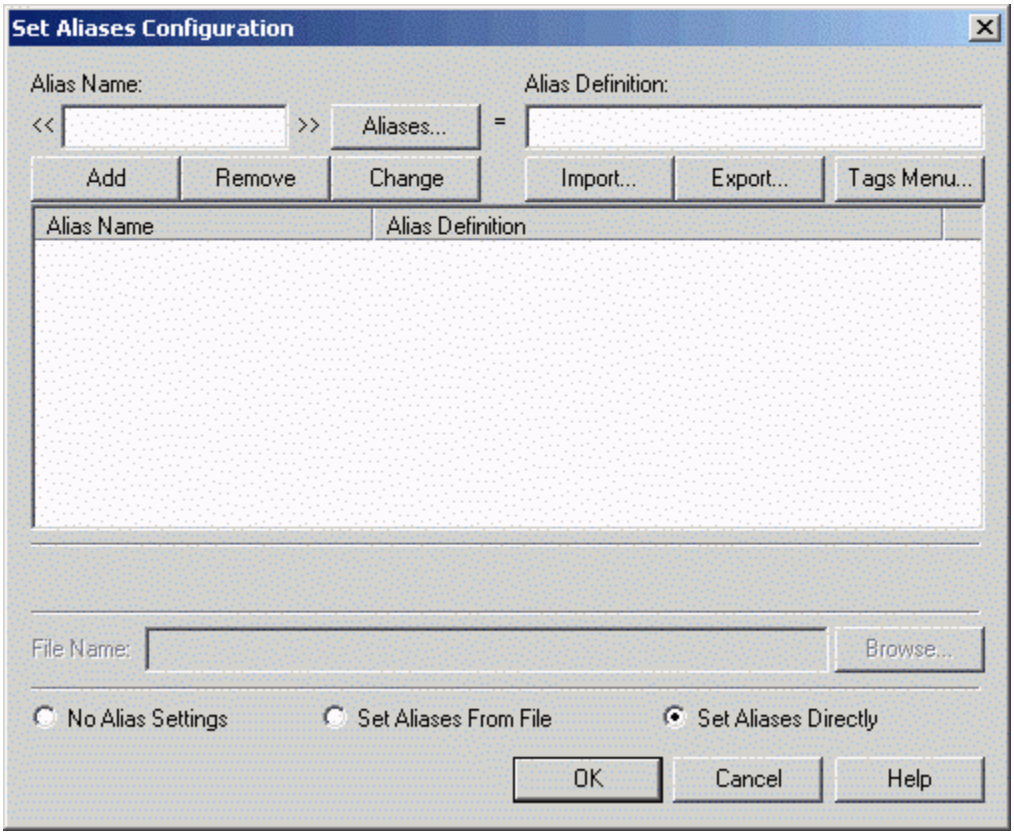
- **Pick action-Load display:** Sets the aliases in the newly loaded display.
- **Pick action-Drag/drop load display:** Sets the aliases in the newly drop-loaded display.
- **Pick action-Popup window:** Sets the aliases in the newly popped-up window.
- **Pick action-Embedded window:** Sets the aliases in the newly embedded window.
- **Pick action-Set aliases:** Sets the specified aliases in the currently open display; can apply to the whole display or a specified object or symbol.
- **Pick action-Alias dialog:** Presents the operator with a dialog, and sets the aliases in the current display based on the selections the operator makes in this dialog; can apply to the whole display or a specified object or symbol.
- **Tab Load Display:** Each tab can include a set of aliases.
- **Launching GraphWorX32 from the command line:** You can specify an alias file as one of the command-line parameters.

To set local aliases, click the **Set Aliases** button on the corresponding pick action tab in the Property Inspector and select **Local Aliases** from the pop-up menu, as shown in the figure below.



Setting Aliases From a Pick Action

This opens the **Set Aliases Configuration** dialog box, shown below. This interface is similar to all of the actions, but it may vary slightly depending upon the action being configured.



Set Aliases Configuration Dialog Box

Set Aliases Parameters

Parameter	Description
Alias Definition	Changes the definitions for all selected alias names.
Tags Menu button	Displays a menu with the following options: **Expression Editor** and **Unified Data Browser.**
No Alias Settings	If this option is selected, no aliases can be specified.
Set Aliases From File	The aliases can come from an external tab-delimited text file These files can be exported from Excel, created in Notepad, or created using GraphWorX32's Alias File Editor. Click the **Browse** button to select a file.
Set Aliases Directly	The aliases to be set can be specified directly in GraphWorX32 using the **Set Aliases Configuration** dialog box.

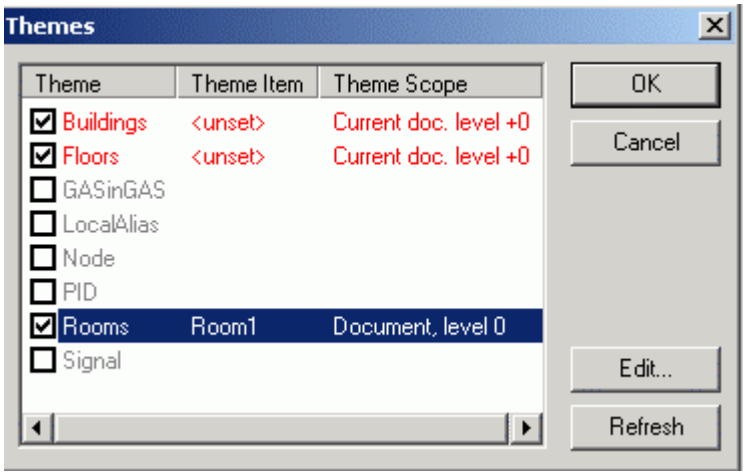
There are also many OLE Automation methods that correspond to many of the above actions. Refer to the **OLE Automation Reference** section for details.

Note: The alias replacement strings are modifiable during runtime via OLE Automation.

You can also specify global aliases by clicking the **Set Aliases** button on the corresponding pick action tab in the Property Inspector and selecting **Global Aliases** from the pop-up menu.

Select a theme from the **Themes** dialog box, as shown in the figure below. Click **OK.**

Note: To edit a theme's properties, select a theme and click the **Edit** button. This opens the **Theme Editor,** where you can specify theme items and the theme scope. For information about editing global alias themes and specifying the theme scope, please see the **Runtime Advanced Settings** section.



Choosing Global Alias Themes

Second-Level Aliasing

Second-level aliasing allows an alias name to be entered as the alias definition. This second alias will be resolved prior to the set-aliases command.

Global Aliasing

GraphWorX32 supports document level, process level, and machine level global global aliasing. A **global alias** is a data string that enables you to reference multiple data sources (e.g. a process point in GraphWorX32) with one unique name. Because you can reference multiple data sources from a single location, global aliasing can reduce the overall number of individual display files you need to create. Global aliasing is integrated into the following GraphWorX32 features:

- Support for visible objects (rectangles, ellipses, lines, etc.)
  - Description
  - Custom command
  - Global Alias Browser for general page
- Support for dynamics (size, location, rotation, etc.)
  - OPC Tag, High Range and Low Range

- Description
- Custom command
- Global Alias Browser for expressions and connections

- Support for text objects (text, button, etc.)
  - Label
  - Global Alias Browser for label and descriptions
- Support for display buttons
  - Display file (supported automatically by inheritance from PICK)
  - Label (supported automatically by inheritance from Text)
  - Global Alias Browser for labels and descriptions
- Support for state fields
  - StateValue
  - StateString
  - Default Value
- Access the Global Alias Browser from the description and language edit boxes for Analog Selector, Animator, Color, ColorAnalog, and Digital Selector
- Flash, Hide, Location, Pick, Process Points, Rotation, Size, Time/Date
- State Fields and connected the code behind the pages.
  - Added browse buttons to Local Variables - InitVal, LoRange, HiRange, DisplayTabs, Windows Properties
- Display Properties
  - Caption
  - Download Value 2x
  - Toggle Value 3x
- Pick action Select GAS (Global Aliasing System) Theme
- Command Line Support: GraphWorX32 allows you to set the initial global alias themes using a command line argument:

  -Themes="MACHINE=<Buildings=Building1"

  The syntax conforms to pick action Select GAS (Global Aliasing System) Theme syntax.

For more information about global aliasing, please see the Global Aliasing Configurator Help documentation.

Language Aliasing

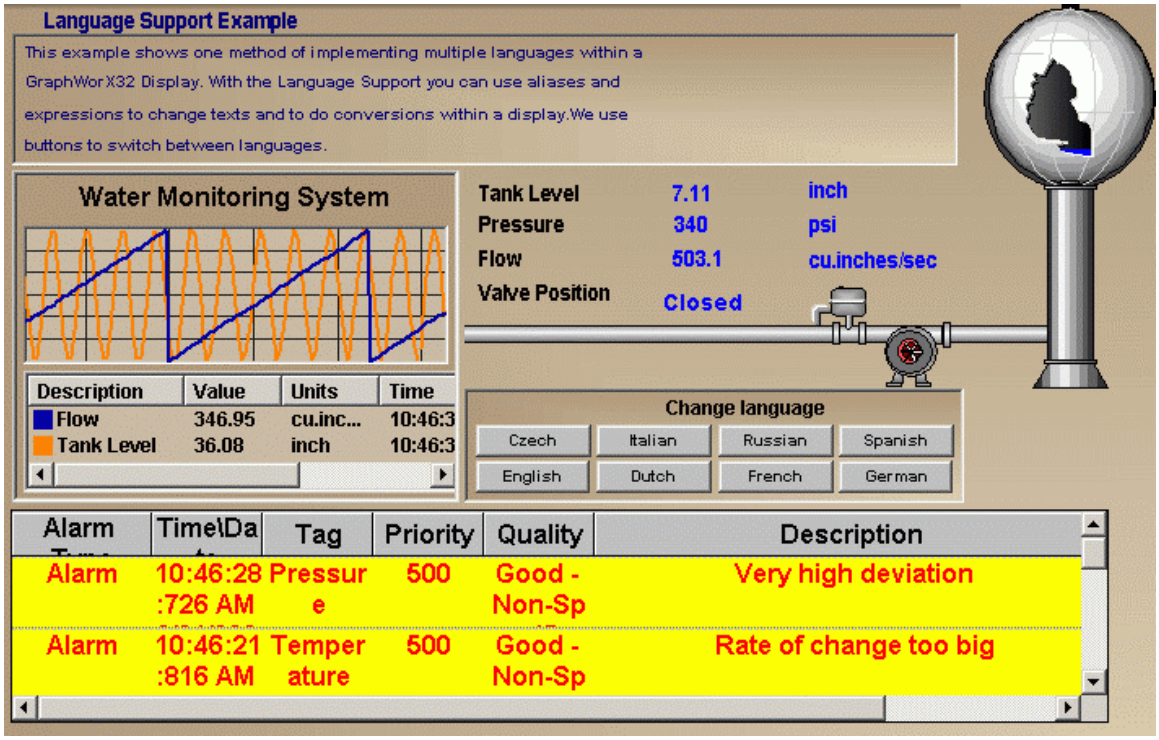
GraphWorX32 has all of its strings saved in the resource file, which can be modified to provide custom versions. In addition, it provides real-time support for loading a resource-only .dll to support an international language. This .dll file is compiled as a standard Win32 .dll. Any time the ICONICS applications settings are switched to a different language, GraphWorX32 will load the corresponding resource .dll.

The Unicode version of GraphWorX32 also supports language aliasing with automatic range and value scaling, as well as unit conversions, formatting, and font selection. For more information about configuring Unicode version language-aliasing support, refer to the **GENESIS32Language Configurator** documentation.

Note: To enhance language-aliasing performance, select the Microsoft Arial Unicode font, which contains all Unicode characters. When you begin configuring in GraphWorX32, you should first select **Application Preferences** from the **Format** menu in GraphWorX32. Click the **Font** button in the **Design** tab of the **Application Preferences** dialog box and select the Arial Unicode MS font. The Arial Unicode MS font must be selected separately within the Properties dialog boxes of each ActiveX component.

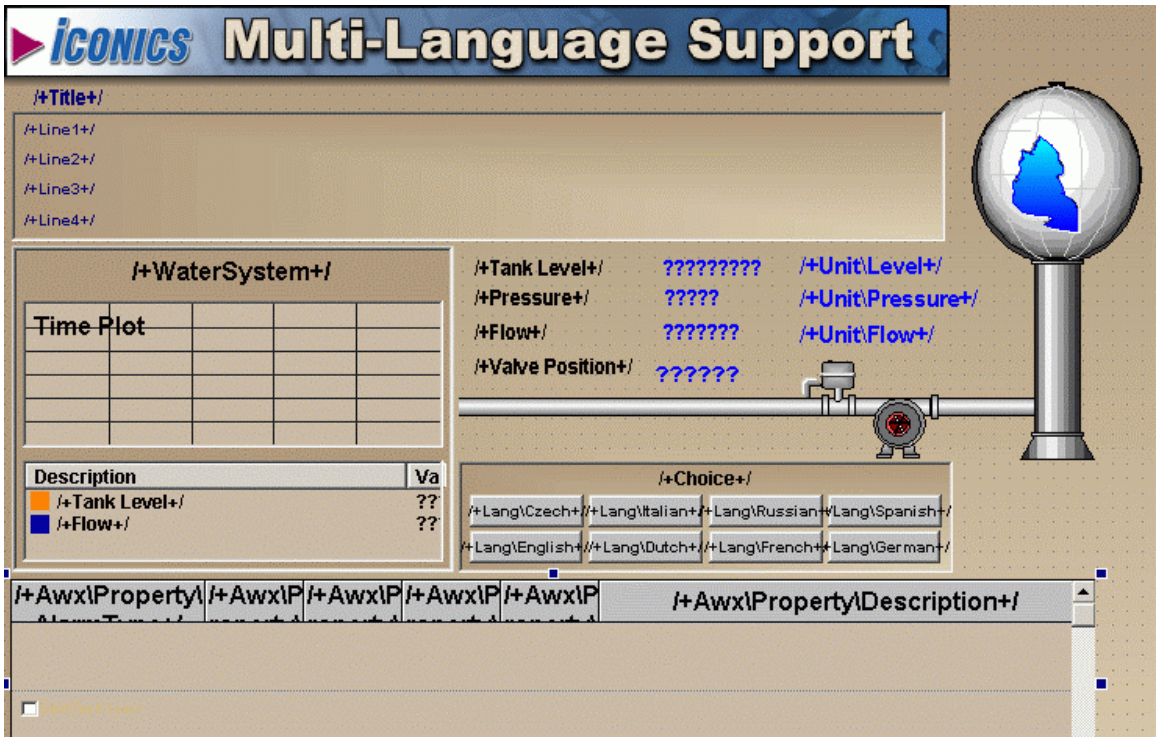
Sample Language Configuration

The Unicode version of GENESIS32 installs a sample Language Server configuration, called **LangSupSample.mdb,** under the Program Files\ICONICS\GENESIS-32\Examples\Languages folder of the GENESIS32 product installation "tree." In addition, this folder includes a sample GraphWorX display, called **LanguagesDemo.gdf,** which includes a sample Viewer configuration. The figure below shows this sample display during runtime mode. Notice that you can switch between languages in this display by simply clicking the buttons in the **Change Language** field.



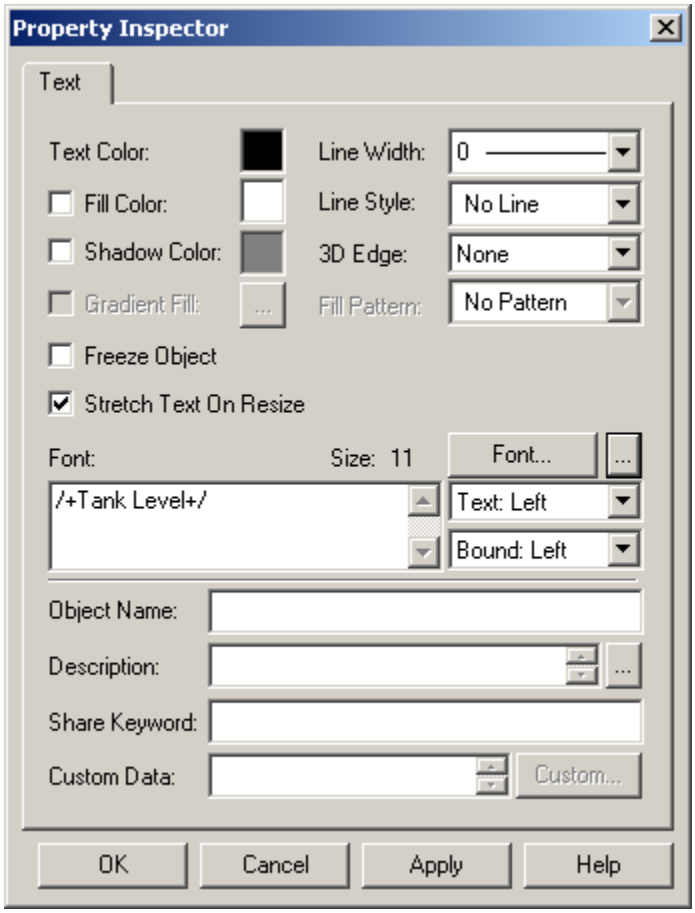
Language-Aliasing Demo in Runtime Mode

The figure below shows this same language-aliasing demo in configuration mode. This display incorporates ActiveX objects from GraphWorX32, TrendWorX32, and AlarmWorX32.



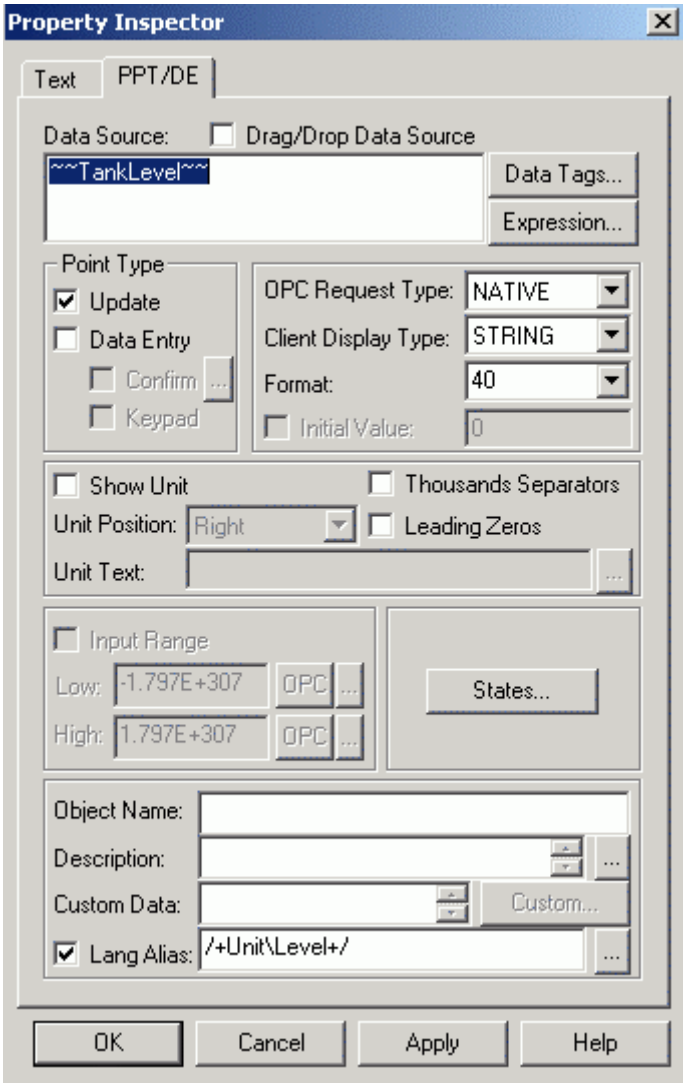
Language-Aliasing Demo in Configuration Mode

For the example above, when you click on the "Tank Level" text box, the **Property Inspector** dialog box opens, as shown in the figure below.



Property Inspector: Text

Similarly, when you click on the process point for "Tank Level," the **Property Inspector** dialog box opens as shown below. The language-aliased string "Unit\Level" appears at the bottom of the **PPT/DE** tab of the **Property Inspector** dialog box in the **Lang Alias** field, as shown in the figure below. Note that the strings within a "/+" and "+/" delimiter pair define a language-aliased string.



Property Inspector: Process Point

Note: The Unicode version language aliasing is independent of the resource .dll selection, which relates mostly to the text in the dialog boxes and other user interface elements.

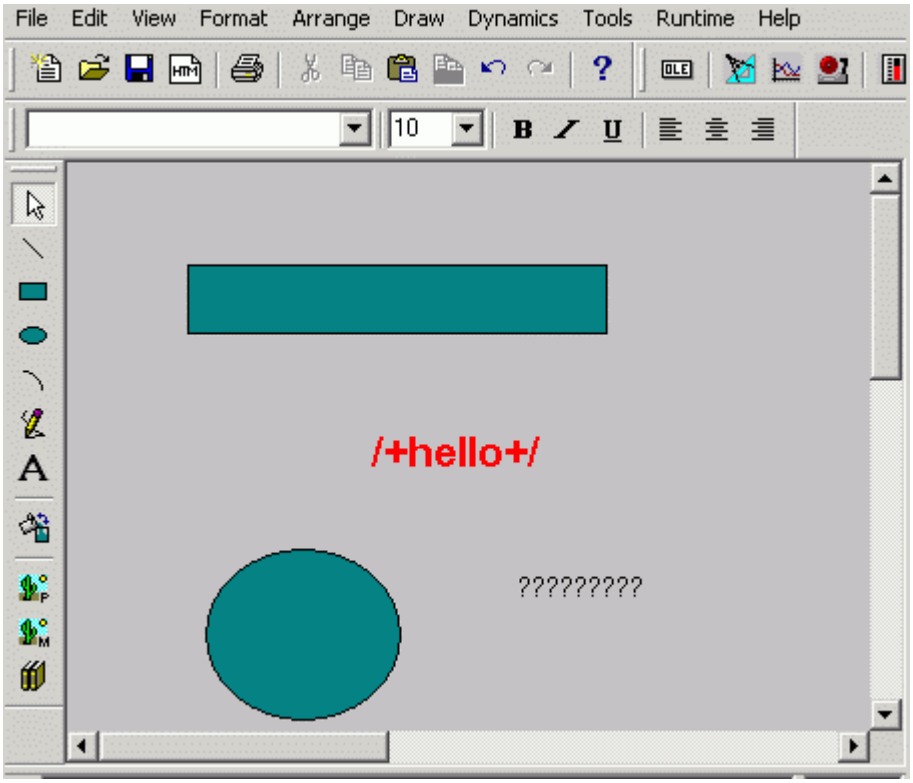
Creating Language-Aliased Strings

Aliases can also be used in the Property Inspector to provide the related aliases for unit scaling, font selection, and date/time translation. During runtime mode, GraphWorX32 interfaces to the ICONICS Language Server, and it tries to resolve the language-aliased strings. The following examples demonstrate how to configure language-aliased strings in GraphWorX32.

Creating a Language-Aliased Text String

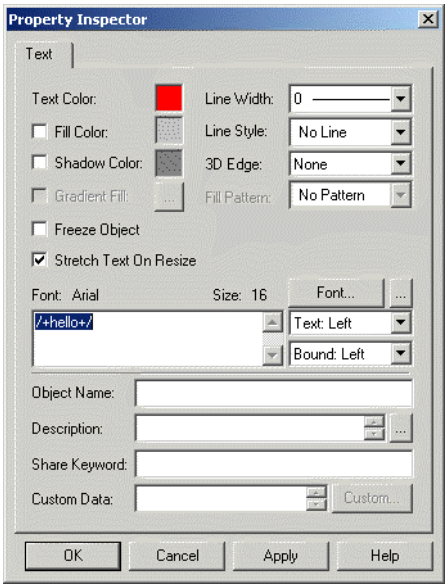
In general, you can use language aliases in GraphWorX32 almost everywhere text is typed. To create a language-aliased text string, do the following:

1. In configuration mode in GraphWorX32, select the **Text** button on the **Draw** toolbar to create a text box. A text box will appear in the display.
2. Type the desired text in the text box. The text should be preceded by the "/+" delimiter and followed by the "+/" delimiter. For example, the word "hello" would be typed as "/+hello+/" in the text box, as shown in the figure below. The delimiters tell the Language Server that the word "hello" is a language-aliased text string.



Text With Delimiters

3. Double-click on the text box to open the Property Inspector dialog box, as shown below. Notice that "/+hello+/" shows up in the text field of the Property Inspector.



Property Inspector With Language-Aliased Text String

4. Open the GENESIS32 Language Configurator from the Windows **Start** menu by selecting **Programs > ICONICS GENESIS32 - Languages** > **Language Configurator.** Configure a new language alias in the language database for the text string you just created. If you do not create an alias for the text, the Language Server will not know how to translate the text. When you create the alias in the Language Configurator, the alias name in the Language Configurator must exactly match that of the text in the GraphWorX32 text box. So in this case, the alias name will be "hello." You can add as many languages as you like to the alias. For each language you add to the alias for "hello," you can enter the translation text for that language. For example, you might use "Hello, how are you?" as the English translation text for the "hello" alias, and you might use "Guten tag" as the German translation for the "hello" alias. Refer to the **Language Configurator** help documentation for detailed information about configuring language aliases.

5. When you have finished configuring the translation for the language alias, enter runtime mode in GraphWorX32. The translated text for "hello" ("Hello, how are you?") now shows up in the GraphWorX32 runtime display, as shown below. Any other language translations that you configure for the alias will also be available in runtime mode. To switch between languages, choose **Select Language** from the **View** menu of GraphWorX32 in runtime mode. This opens the **Language Selector** dialog box. Choose the language you would like to have displayed. In this case, if you choose German, "Guten tag" will show up in the display.

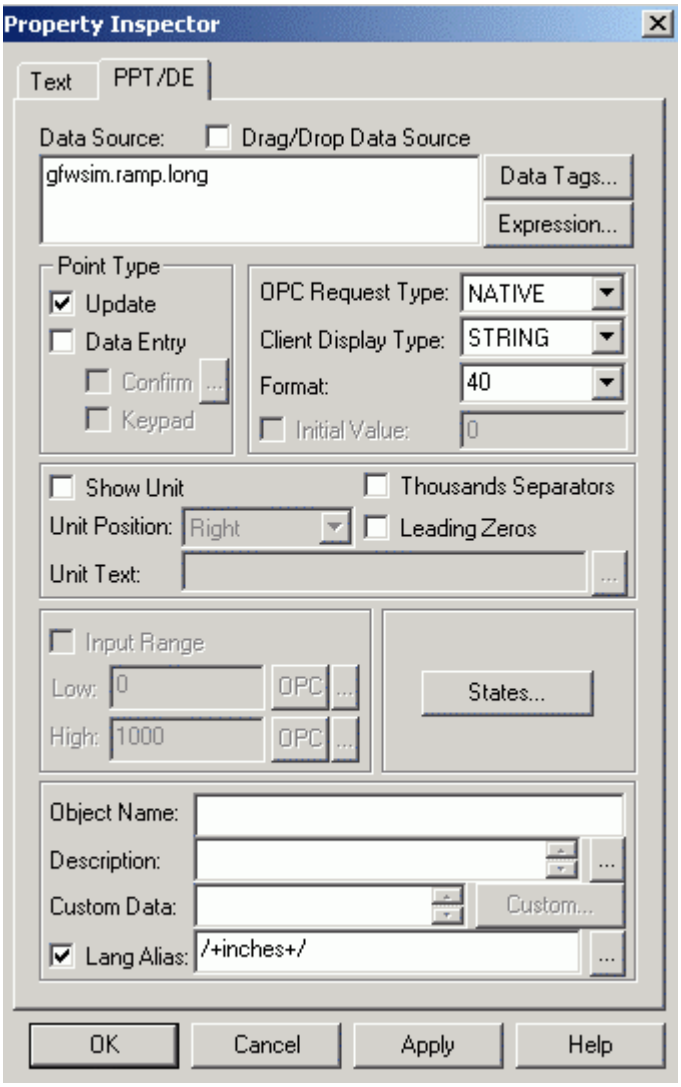


Language Translation in Runtime Mode

Creating Other Language-Aliased Strings

Language aliases can also be used in the Property Inspector to provide the related aliases for unit scaling, font selection, and date/time translation. For example, let's assume that you have configured a language-aliased text string in GraphWorX32 for "/+inches+/" using the method described above. Using the ICONICS Language Configurator, you can create an "inches" alias for the text string. Let's suppose that, as in the example above, you want to enter English and German translations for the "inches" alias. For example, you might use "inches" as the English translation text for the "inches" alias, and you might use "cm" (centimeter) as the German translation for the "text" alias.

1. If you want to associate the "inches" alias with a dynamic action in GraphWorX32, such as a process point, you need to tell the Language Server how to convert between inches and centimeters. Thus, you must also associate the alias with an expression that defines the conversion. First you need to create a new expression using the GENESIS32 Language Configurator. Then you need to add this expression to the "inches" alias. For more details about how to add an expression, refer to the Language Configurator help documentation.
2. Once you have added the unit conversion expression to the "inches" alias, you can now associate the alias with a process point. In GraphWorX32, select **Intrinsics > Process Point** from the **Dynamics** menu. This adds a process point to the display.
3. Click on the GraphWorX32 display to open the Property Inspector dialog box. Click on the **PPT/DE** tab of the Property Inspector. Select a data source. In this example, we will use the "gfwsim.ramp.long" simulation variable, as shown in the figure below.



Process Point With Language-Aliased String

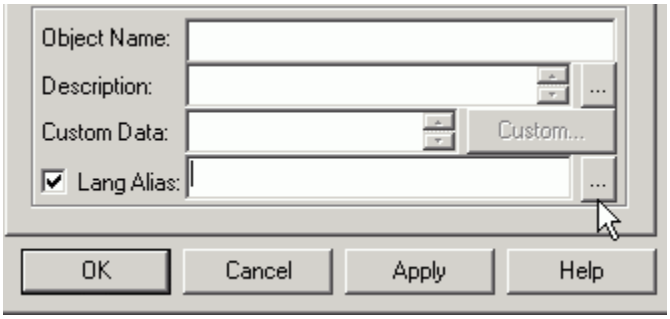
4. Check the Lang. Alias box at the bottom of the Property Inspector, as shown below. This enables the Language Alias field.
5. Enter the "/+inches+/" language alias string in the Language Alias field, as shown below. Click Apply. Now the process point is associated with the "inches" alias.

   Note: You can also select a language alias from the Language Alias Browser by clicking the ... button next to the Lang Alias box and selecting Language Aliases. For more information about using the Language Alias Browser, please see the Selecting Language Aliases From the Language Alias Browser section.
6. When runtime mode is activated, you will see the dynamic process-point values for "gfwsim.ramp.long." To switch between languages, choose Select Language from the View menu of GraphWorX32 in runtime mode. This opens the Language Selector dialog box. Choose the language you would like to have displayed. If English is selected, the process-point values will be in *inches.* If German is selected, the values will be in *centimeters (cm).*

Selecting Language Aliases From the Unified Data Browser

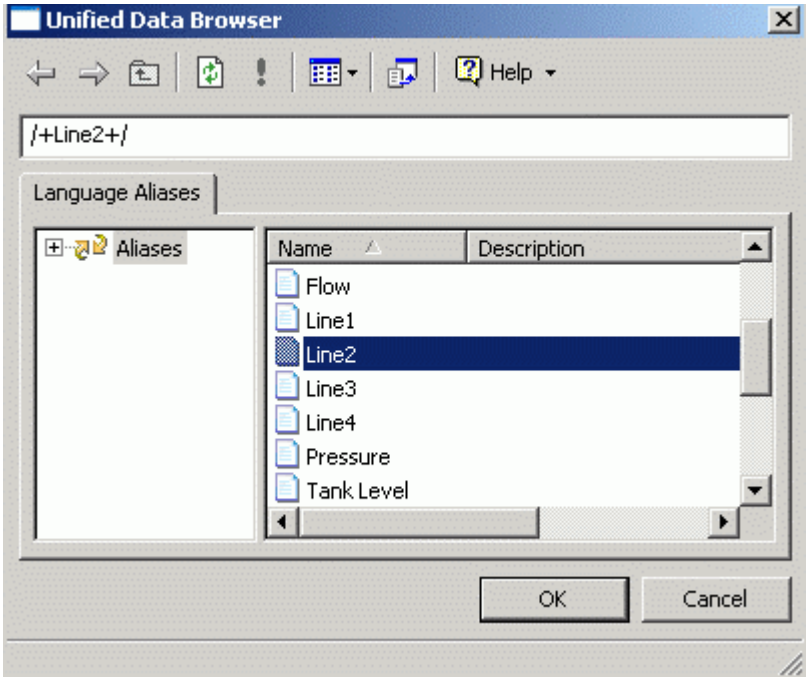
When specifying a language alias in the Property Inspector, you can also select a language alias from the Language Alias Browser, which includes all language aliases in the language database. This eliminates the need to manually type in the alias name.

1. In the Property Inspector (e.g. for a process point), check the **Lang Alias** check box and click the **...** button next and select a language alias from the **Unified Data Browser,** as shown in the figure below.



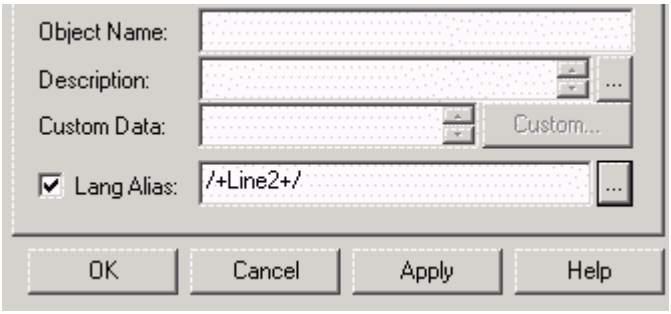
Opening the Language Alias Browser

2. The **Language Alias Browser** opens, as shown in the figure below. The browser includes all languages aliases in the language database. All language aliases that are configured in the Language Configurator are conveniently available to choose from inside the browser. The tree control of the Language Configurator is mimicked in the tree control of the Language Alias Browser. Select a language alias by double-clicking the alias name (e.g. "Line 2" in the figure below). The alias name appears at the top of the browser, which automatically adds the /+ and +/ delimiters to the alias name. Click the **OK** button.



Selecting a Language Alias From the Unified Data Browser

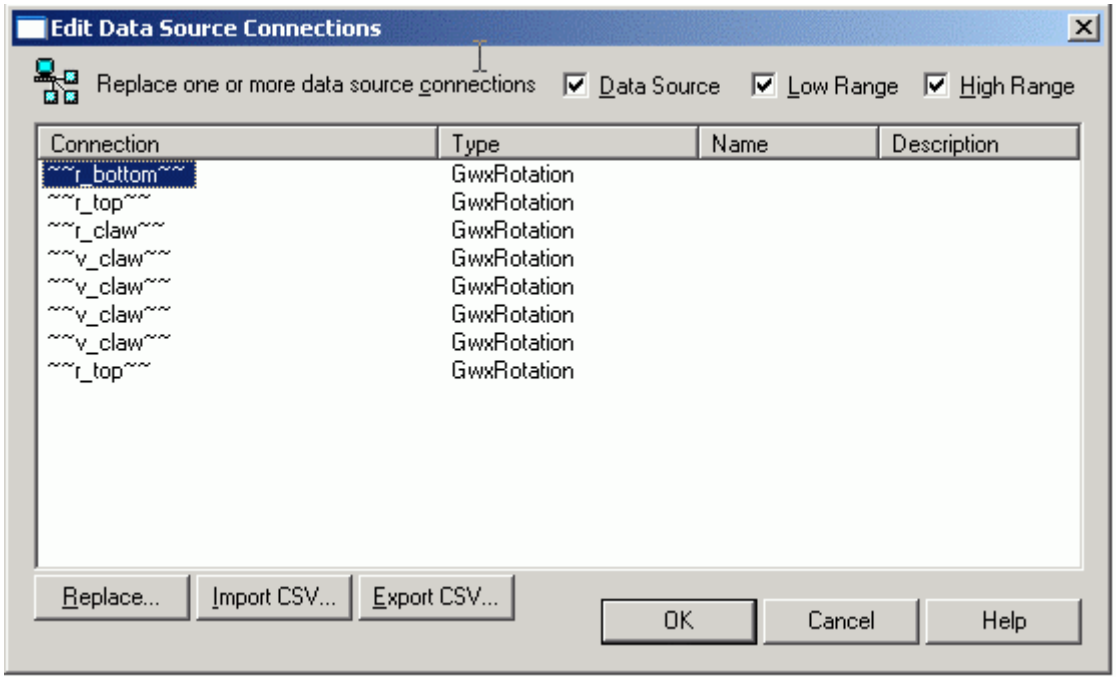
3. The language alias (with delimiters) is added to the **Lang Alias** field of the Property Inspector dialog box, as shown in the figure below.



Language Alias Added to Property Inspector

Editing Data Source Connections

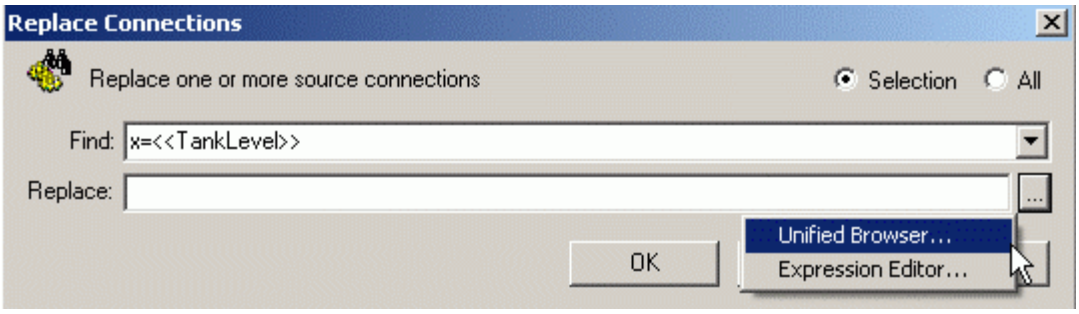
Selecting **Edit Connections** from the **Dynamics** menu opens the **Edit Data Source Connection** dialog box, shown in the figure below, which simplifies the connection of GraphWorX32 symbols to OPC data.



Editing the Data Source Connection

Replacing Connections

Clicking the **Replace** button opens the **Replace Connections** dialog box, as shown in the figure below. This allows you to run a search and replace operation for one or more connections on one or more symbols in the current display. The replace capability supports wildcard replacement, similar to Find / Replace feature.



Replace Connections Dialog Box

Import and Export

In addition, the tags can be automatically connected to symbols by importing a *.csv file and using it for auto connections. Click the **Import CSV** button to browse for a .csv file.

The existing connections can also be exported to *.csv file for a further use. Click the **Export CSV** button to browse for a .csv file.

Chapter 8: Dynamic Connections

This chapter explains how to make dynamic connections from display objects to points coming from various OPC data servers. The objects are animated and controlled by the values of those data points. You can make multiple dynamic connections to a single object. For example, you can create an object that changes in color and size based on values from two different point connections.

You create dynamic connections with functions contained in the **Dynamics** menu. Types of dynamic connections include flash, color, process point (PPT), data entry, size, location, analog and digital selector, animator, hide, and rotation.

Contents

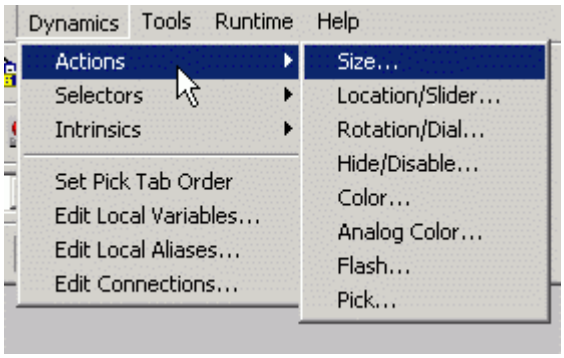
Data Source
Action Dynamics
Selector Dynamics
Intrinsic Dynamics
Customizing
Confirmation Message Builder
Data Mining

Data Source

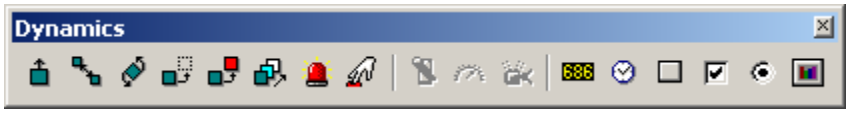
In every dynamic dialog there is a data source. Refer to the **Data Connections** sections for more information.

Action Dynamics

Dynamic actions connections apply specific actions to objects in a display based on data-point values. You can create these connections through the **Actions** submenu on the **Dynamics** menu, as shown in the figure below, or by clicking the appropriate button on the **Dynamics** toolbar, shown below.



Dynamics - Actions Menu



Dynamics Toolbar

Size Connection
Location/Slider
Rotation/Dial
Hide/Disable
Digital Color
Analog Color
Flash
Pick Actions

Size Connection

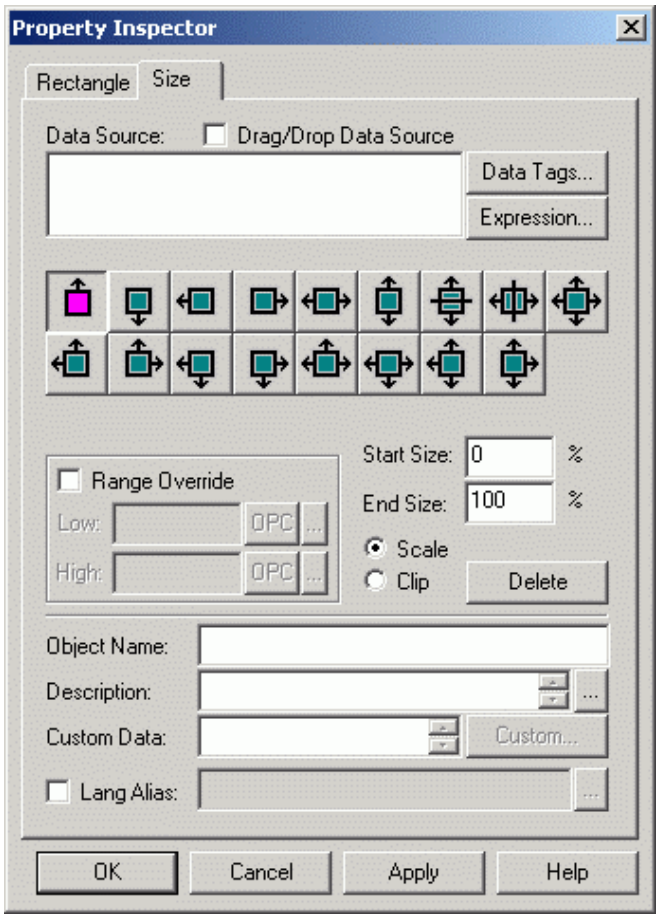


Using the **Size** button on the **Dynamics** toolbar, you can define an object that changes size based on the real-time value of its connected data point. The object automatically sizes to scale for incoming data points, such that a 100 percent scale equals the full size of the drawn object. You can override the size by defining range override values.

During runtime mode, the **Size** connection resizes the object in proportion to the percentage of the high and low ranges of the connected data point.

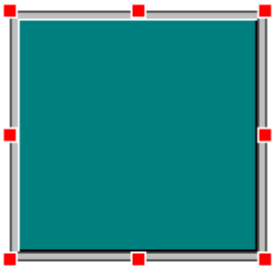
To establish a size connection:

1. Select an object in the display.
2. Select **Actions > Size** from the **Dynamics** menu, or click the **Size** button on the **Dynamics** toolbar. The **Size** tab of the Property Inspector for the object opens, as shown below.
3. Establish a data connection.
4. Select a size action from the size action parameters listed in the table below.



Property Inspector Dialog Box: Size Tab

5. Click **OK.** A highlighted box with red handles surrounds the object indicating a dynamic connection, as shown below.



Object With Dynamic Connection Established

Size Action Parameters

Parameter	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.
Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Size Toolbar	Specifies the direction in which to size the object when the appropriate button is clicked.
Vertical Buttons	Expand the object from the bottom up or top down.
Horizontal Buttons	Expand the object from the left or right.
Bias Bars Buttons	Expand the object from the center above and below the bias line, or to the left and right of the bias line.
Two Directional Button	Expands the object outward from the center.
Three Directional Button	Expands the object in three directions outward from the center.
Range Override	Activates an operating range other than the default range for the data point. Any dynamic based on an analog value, such as a process point or analog selector, will have a range override.
Start Size/End Size	Specifies (as a percentage) the starting and ending size of an object that has changing dimensions. The size does not have to be an integer.
Scale	Scales the object in proportion to the incoming signal.
Clip	Reveals the object.
Delete	Deletes this dynamic. This is only available on actions shown on the Dynamic Actions toolbar.
Object Name	Identifies the object for OLE Automation. A name specified for this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or

	Language Aliases from the alias browsers by clicking the ... button to the right of the **Description** field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** below for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

Location/Slider

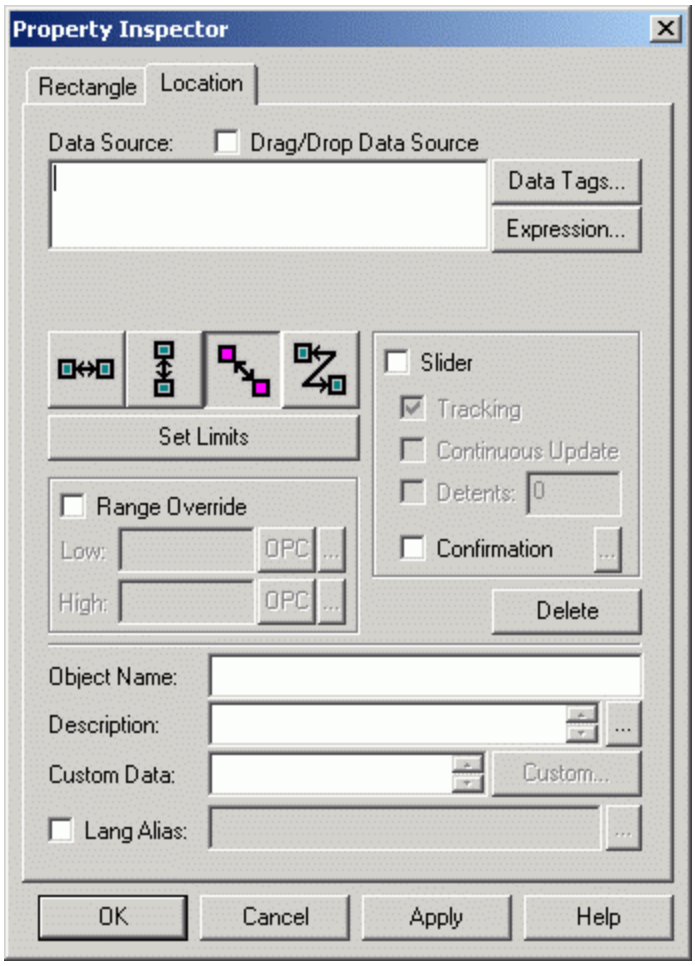


The **Location/Slider** dynamic on the **Dynamics** toolbar enables you to change the location of an object based on the value of a data point. During runtime mode, the object moves within a specified distance (horizontally, vertically, diagonally, or along a specified path) based on the value of the connected data point.

The **Slider** function emulates a mechanical slider control. During runtime mode, you can move the slider knob to change the value of the connected data point.

To make a location connection:

1. Select an object in the display.
2. Select **Actions > Location/Slider** from the **Dynamics** menu, or click the **Location/Slider** button on the **Dynamics** toolbar. The **Location** tab of the Property Inspector for the object opens, as shown below.



Property Inspector Dialog Box: Location Tab

3. Specify the data source tag and select the location/slider action parameters that are described in the table below.
4. Click **OK.** Red handles appear around the object, indicating that a dynamic connection has been made.

Location Action Parameters

Parameter	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.
Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Side to Side Button	Object moves from side to side.
Up and Down Button	Object moves up and down.
Diagonal Button	Object moves diagonally.
Free-Form Path Button	Object moves along a user-specified free-form path.
Range Override	Activates an operating range other than the default range for the data point.
Set limits	Defines an area within which the object will move. A gray frame around the object indicates that the object in runtime will move within that area.
Slider	The slider allows you to hold the object with the mouse and move the object.
Tracking	This is part of the slider action. If this option is checked, the object will move based on the connected data value. If it is not checked, the object will remain stationary, unless you move it manually.
Continuous Update	The **Continuous Update** causes continuous writing of the current value to the OPC server (i.e. on each movement of the slider dynamics) as quickly as possible. An OPC server on a fast connection will receive more updates than an OPC server on a slow connection. When this check box is checked, the **Confirmation** check box is disabled.

Delete	Deletes this dynamic.
Detents	Also a part of the slider action. The range of the variable, tag, or expression that the slider is controlling is divided into *N* steps including the high and low range, where *N* is the value entered in the **Detents** field.
Confirmation	Defines a confirmation message, which is a message the GraphWorX32 prompts every time the user writes a value. Click the **…** button to open the **Confirmation Message Builder,** which helps you create confirmation messages. For more information, please see the Confirmation Message Builder section below.
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

Rotation/Dial

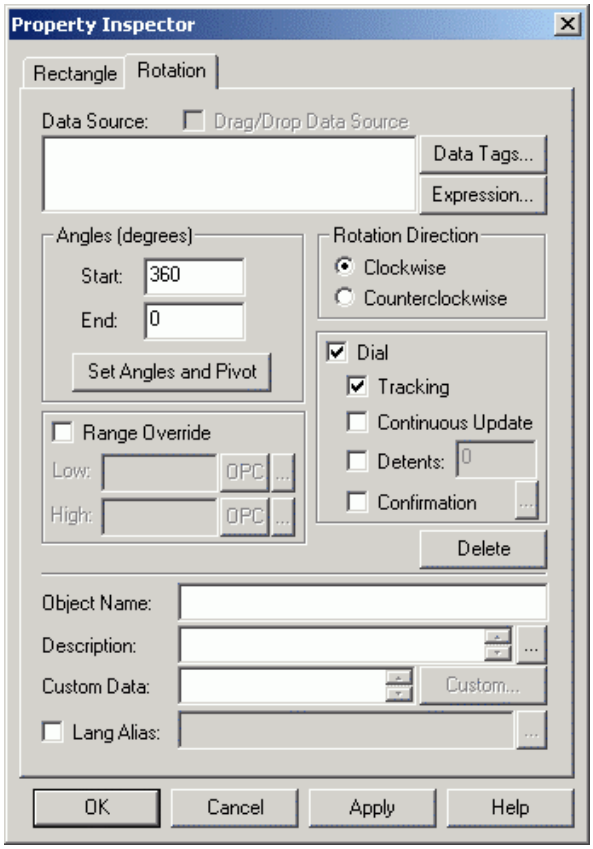


The **Rotation/Dial** dynamic on the **Dynamics** toolbar enables you to rotate an object based on the connected data point value. You define the rotation pivot coordinates. This function is useful when you want to graphically display information on dials or meters in your display.

To define rotation for an object:

1. Select the object.
2. Select **Actions > Rotation/Dial** from the **Dynamics** menu, or click the **Rotation/Dial** button on the **Dynamics** toolbar.
3. The **Rotation** tab of the Property Inspector for the object opens, as shown below.
4. Select the desired parameters described in the table below, and then click **OK.**

Note: It is possible to freely rotate any image in GraphWorX32 both in configuration mode using the “free rotate” tool and in runtime mode using the rotation dynamic.



Property Inspector Dialog Box: Rotation Tab

Rotation Parameters

Parameter	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.
Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Start	Sets the angle at which the rotation starts.
End	Sets the angle at which the rotation ends.
Set Angles and Pivot	Sets the angle of rotation and defines the coordinates of the pivot point of the object from which it rotates.
Rotation Direction	Specifies a rotation direction (clockwise or counterclockwise).
Dial	Similar to the Slider function described for the Location action. However, instead of allowing you to manually move the object up and down, **Dial** allows you to manually rotate the object.
Tracking	This is part of the dial action. If this option is checked, the object moves based on the connected data value. If it is not checked, the object remains stationary, unless you move it manually.
Continuous Update	The **Continuous Update** causes continuous writing of the current value to the OPC server (i.e. on each movement of the dial dynamics) as quickly as possible. An OPC server on a fast connection will receive more updates than an OPC server on a slow connection. When this check box is checked, the **Confirmation** check box is disabled.
Detents	Also a part of the dial action. The range of the variable, tag, or expression that the slider is controlling is divided into *N* steps including the high and low range, where *N* is the value entered

	in the **Detents** field.
Confirmation	Defines a confirmation message, which is a message the GraphWorX32 prompts every time the user writes a value. Click the **…** button to open the **Confirmation Message Builder,** which helps you create confirmation messages. For more information, please see the Confirmation Message Builder section below.
Range Override	Activates an operating range other than the default range for the data point.
Delete	Deletes this dynamic.
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

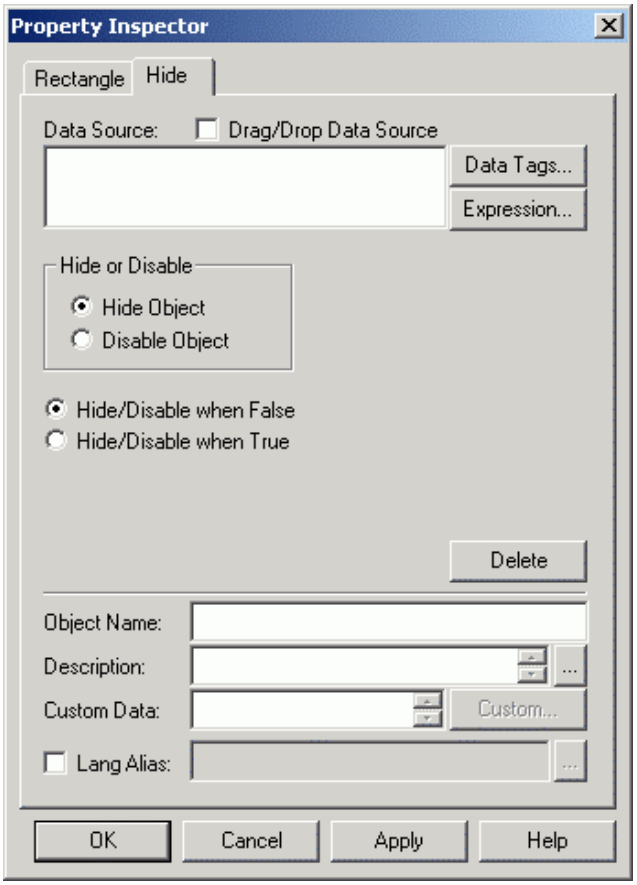
Hide/Disable



The **Hide/Disable** dynamic on the **Dynamics** toolbar causes objects to become hidden when the value of the digital data point to which it is connected goes either true (logical 1) or false (logical 0) as you configure it.

To make a hide connection:

1. Select an object in the display.
2. Select **Actions > Hide/Disable** from the **Dynamics** menu, or click the **Hide/Disable** button on the **Dynamics** toolbar.
3. The **Hide** tab of the Property Inspector for the object opens, as shown below.
4. Establish a digital data-point connection.
5. Select to hide or disable the object.
6. Select to hide the object when the data-point value is true or when the data-point value is false. Then click **OK.**



Property Inspector Dialog Box: Hide Tab

Hide Parameters

Parameter	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.
Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Hide Object	Hides the selected object in the display.
Disable Object	Disables the selected object in the display.
Hide/Disable When False	Causes the objects to become hidden when the value of the digital data point to which it is connected goes to false (logical 0).
Hide/Disable When True	Causes the objects to become disabled when the value of the digital data point to which it is connected goes to true (logical 1).
Delete	Deletes this dynamic.
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the

	... button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

Digital Color

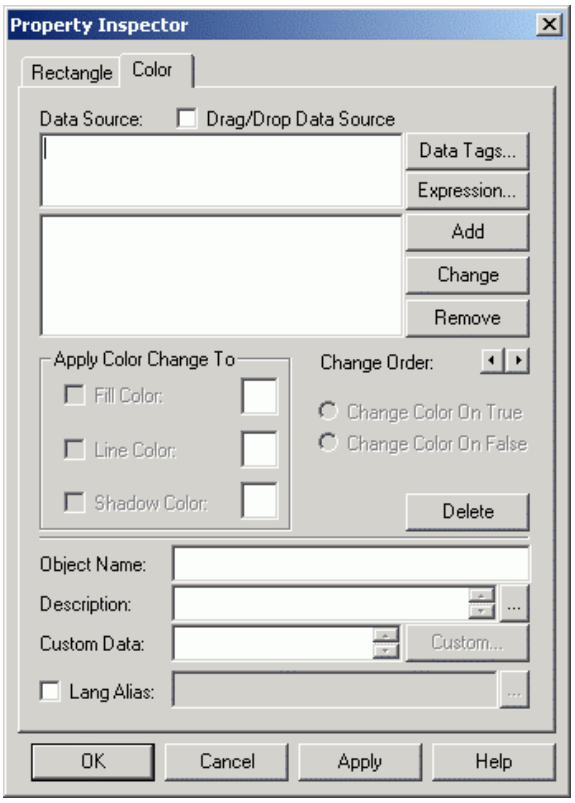


The **Digital Color** dynamic on the **Dynamics** toolbar allows the color of the object to change based on a digital event. The color connections are prioritized in the order in which they are created. Therefore, if two data points are true at the same time, the higher-priority color connection takes precedence. If none of the connected data points is true, the object is displayed in its original color.

The **Color** connection is use to indicate states such as alarm conditions and temperature changes, or to notify operators of the flow in a pipe.

To create a digital color connection:

1. Select an object in the display.
2. Select **Actions > Color** from the **Dynamics** menu, or click the **Color** button on the **Dynamics** toolbar. The **Color** tab of the Property Inspector for the object opens, as shown below.
3. Select the desired parameters, and then click **OK.**



Property Inspector Dialog Box: Color Tab

Color Connection Parameters

Parameter	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.
Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Add Button	Adds the string currently in the data source edit control to the list of data sources. Click the arrows next to **Change Order** to sort the data sources.
Change Button	Changes existing data connections. The currently selected item in the data source list box is changed to the string that is currently in the data source edit control.
Remove Button	Removes existing data connections from the data source list box.
Apply Color Change To	These items vary depending on which source string is selected.
Fill Color	If checked, changes the fill color of the object. Clicking on the box to the right of the **Fill Color** check box opens the color palette.
Line Color	Changes the color of the object's border. Clicking on the box to the right of the **Line Color** check box opens the color palette.
Shadow Color	Changes the color of the object's shadow. Clicking on the box to the right of the **Shadow Color** check box opens the color palette.
Change Color on True/False	Changes the color of the object when the connected data point goes true (logical 1) or false (logical 0).
Delete	Deletes this dynamic.

Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

Analog Color

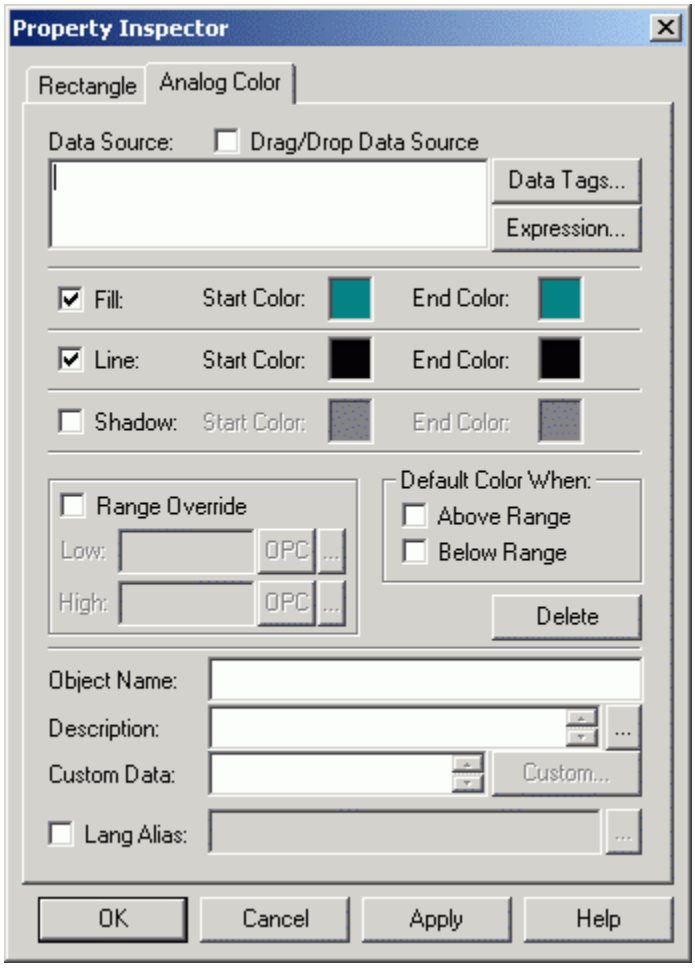


The **Analog Color** dynamic on the **Dynamics** toolbar changes the start color and end color based on an analog signal.

Note: You can attach multiple analog color dynamics to shift between more than two colors.

To create an analog color connection:

1. Select an object in the display.
2. Select **Actions > Analog Color** from the **Dynamics** menu. The **Analog Color** tab of the Property Inspector for the object opens, as shown below.
3. Select the desired parameters described in the table below, and then click **OK.**



Property Inspector Dialog Box: Analog Color Tab

Analog Color Parameters

Parameters	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.

Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Fill Color - Start/End	Changes the starting and ending fill colors of the object. Clicking on the **Start Color** and **End Color** boxes opens the Color Palette.
Start Color	Changes the starting fill color of the object. This is the color that will appear when the analog signal is at its lowest value.
End Color	Changes the ending fill color of the object. This is the color that will appear when the analog signal is at its highest value. When the analog signal is between its minimum and maximum values, the color will be a mix of the start and end colors.
Line Color - Start/End	Changes the starting and ending line colors of the object.
Shadow Color - Start/End	Changes the starting and ending shadow colors of the object.
Range Override: Low/High	Activates an operating range other than the default with specific options of choosing low range and/or high range.
Default Color when Above Range/Below Range	Sets the default color when above or below the specified range. The default color is the color of the object to which this dynamic is attached.
Delete	Deletes this dynamic.
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

Flash

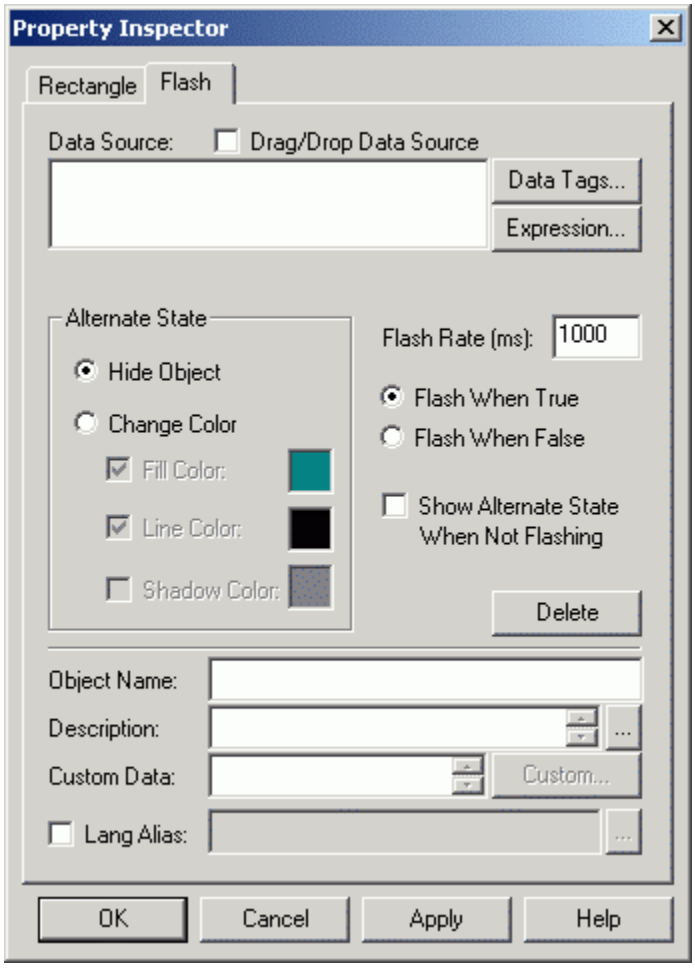


The **Flash** dynamic on the **Dynamics** toolbar causes an object with a digital connection to blink on and off or to flash between two colors during runtime mode when the connected digital value goes true (logical 1) or false (logical 0).

The default operation is for the object to flash between configured colors. The Property Inspector shows the default as on/off. When the data-point value is false (logical 0), the object displays in the normally on/normally off setting. When the value is true (logical 1), the object blinks between the colors or between visible and hidden. The rate at which the object blinks is determined by the **Flash Rate** value.

To make a flashing connection:

1. Select an object in the display.
2. Select **Actions > Flash** from the **Dynamics** menu, or click the **Flash** button on the **Dynamics** toolbar.
3. The **Flash** tab of the Property Inspector for the object opens, as shown below.
4. Establish a data connection and select the desired parameters described in the table below.
5. Click **OK** when you are finished.



Property Inspector Dialog Box: Flash Tab

Flashing Parameters

Parameter	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.

Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Flash Rate	The frequency of the flash in milliseconds. The minimum flash value is 1000 ms.
Flash When True	Flashes when the data-point value is true (logical 1). The object blinks between the colors or between visible and hidden.
Flash When False	Flashes when the data-point value is false (logical 0).
Hide Object	Selects the alternate state as a hidden object.
Change Color	Selects the alternate state as a color change.
Fill Color	Selects the fill color for the object. Clicking on the box at right opens the Color Palette.
Line Color	Selects the line color for the object. Clicking on the box at right opens the Color Palette.
Shadow Color	Selects the shadow color for the object. Clicking on the box at right opens the Color Palette.
Show Alternate State When Not Flashing	When not flashing, the object is shown in the specified alternate color, or the object is hidden if that is the chosen alternate state.
Delete	Deletes this dynamic.
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

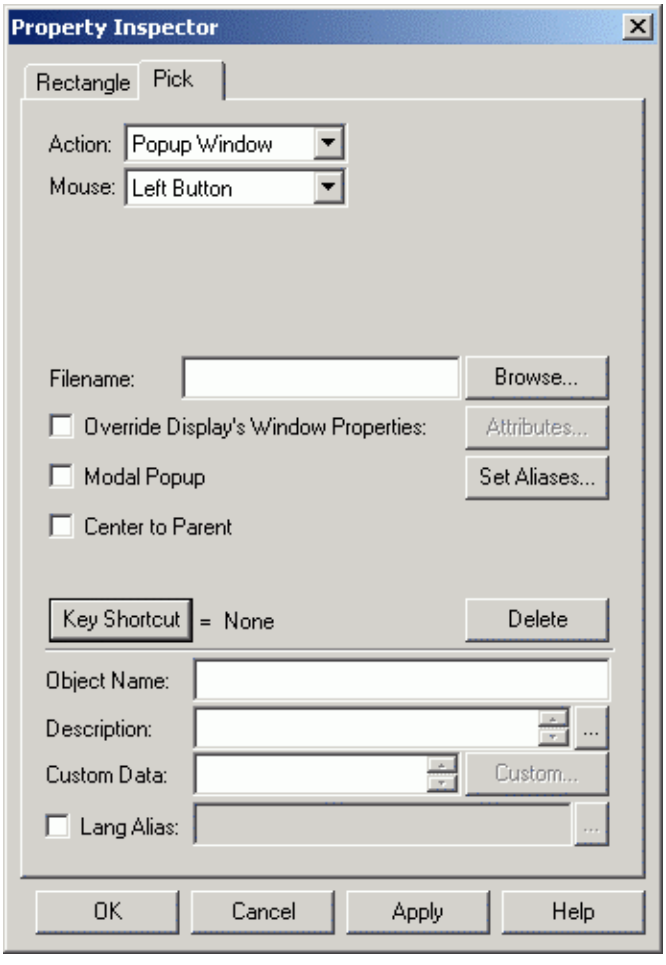
Pick Actions



The **Pick Action** function allows you to configure an object such that it performs an action when the object is clicked during runtime mode. Such actions include loading displays, popping up windows, launching applications, and downloading values.

To make a pick action connection:

1. Select an object in the display.
2. Select **Actions > Pick** from the **Dynamics** menu, or click the **Pick** button on the **Dynamics** toolbar.
3. The **Pick** tab of the Property Inspector for the object opens, as shown below.
4. Select an action type from the drop-down list in the **Action** field. The appearance of the **Pick** tab varies depending upon which action is selected from the drop-down list.



Property Inspector Dialog Box: Pick Action Tab

Defining Key Shortcuts

Clicking the **Key Shortcut** button opens the **Define Key Shortcut** dialog box, shown below. Enter a key and/or choose a modifier, and then click **OK.** When the display is changed to runtime mode, the operator can perform the pick action by pressing the designated shortcut key.



Define Key Shortcut Dialog Box

Parameters for Pick Actions

Pick actions are listed in the **Action** field in the **Pick** tab of the Property Inspector. Although the parameters for each pick action vary, many of the various pick actions available for pick dynamics have common properties, which are described in the table below.

Pick Action Parameters

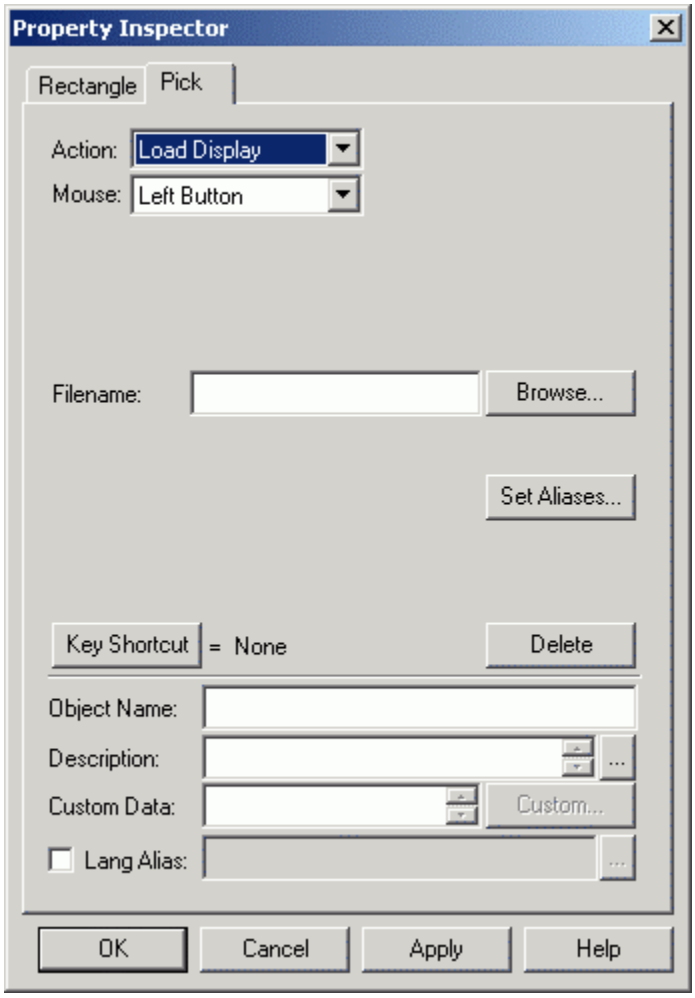
Parameter	Description
Mouse	Lets you choose which mouse button will activate the pick during runtime mode (can be left, middle, or right mouse button).
Delete	Deletes this dynamic.
Type	There are two different "picks": Object pick and three styles of buttons: Normal, Check Box, and Radio.
Check Button	This is a check box. Most of the parameters are the same as the Normal button.
Radio Button	When this pick is selected, any other picks that are below the same group are automatically deselected (i.e. they are mutually exclusive).
Initial state	Available only for check button and radio button types. Lets you choose an initial state, which can be up or down.
Group	Available only for the radio button type, this specifies the group name to which the radio button belongs.
Execution Trigger	Selects one of three options for executing the action; On Down, While Down, or On Up.
On Down	Executes the action when the mouse button is clicked down.
On Up	Executes the action when the mouse button is released.

While Down	Executes the action while the mouse button is held down. Lets you specify an interval in milliseconds.
Key Shortcut	Defines a shortcut key for the pick action.
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

Action - Load Display
Action - Drag/Drop Load
Action - Display Back and Display Forward
Action - Popup Window
Action - Embedded Window
Action - Launch Application or Document
Launch Web Page
Action - Close Window
Action - Toggle Value
Action - Download Value
Action - Set Local Aliases
Action - Local Alias Dialog
Action - Run Script
Action - Custom Command
Action - Popup Menu
Action - Select Language
Actions - Layers
Action - Select Global Alias Theme
Action - Run Report
Action - Run Transaction
Action - Database Update
Action - VCR Panel Show/Hide

Action - Load Display

The **Load Display** pick action in the **Pick** tab of the Property Inspector loads the display in runtime mode. You can also configure aliases by clicking the **Set Aliases** button.



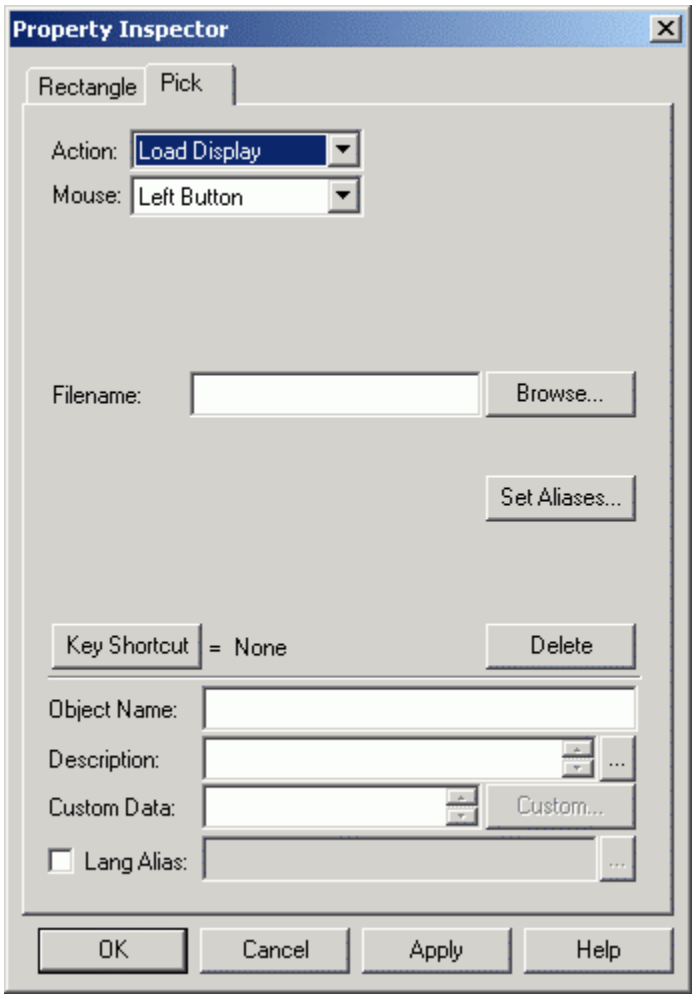
Property Inspector Dialog Box: Load Display Action

Load Display Parameters

Parameter	Description
File Name	Click the **Browse** button to pick a file name (the display file to be loaded during runtime).

Action - Drag/Drop Load

The **Drag/Drop Load** pick action in the **Pick** tab of the Property Inspector allows you to drag-and-drop a load display to another GraphWorX32 window (not on itself or in a parent window). This provides an easy mechanism for choosing into which window the display gets loaded.



Property Inspector Dialog Box: Drag/Drop Load Action

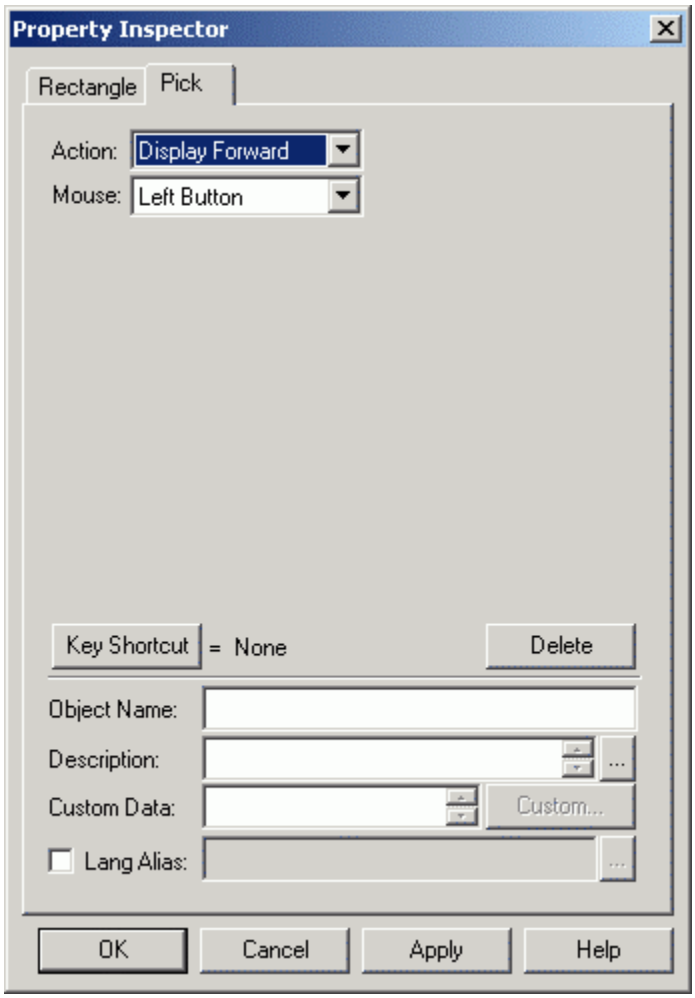
Drag/Drop Load Parameters

Parameter	Description
File Name	Click the **Browse** button to pick a file name (the display file to be loaded during runtime).

Action - Display Back and Display Forward

GraphWorX32 maintains a history of the last 50 displays. You can navigate through the display history using the pick actions **Display Back** and **Display Forward** in the **Pick** tab of the Property Inspector. These commands are also available as pick actions and as OLE Automation methods. The Display History feature is similar to Microsoft Internet Explorer's back/forward feature.

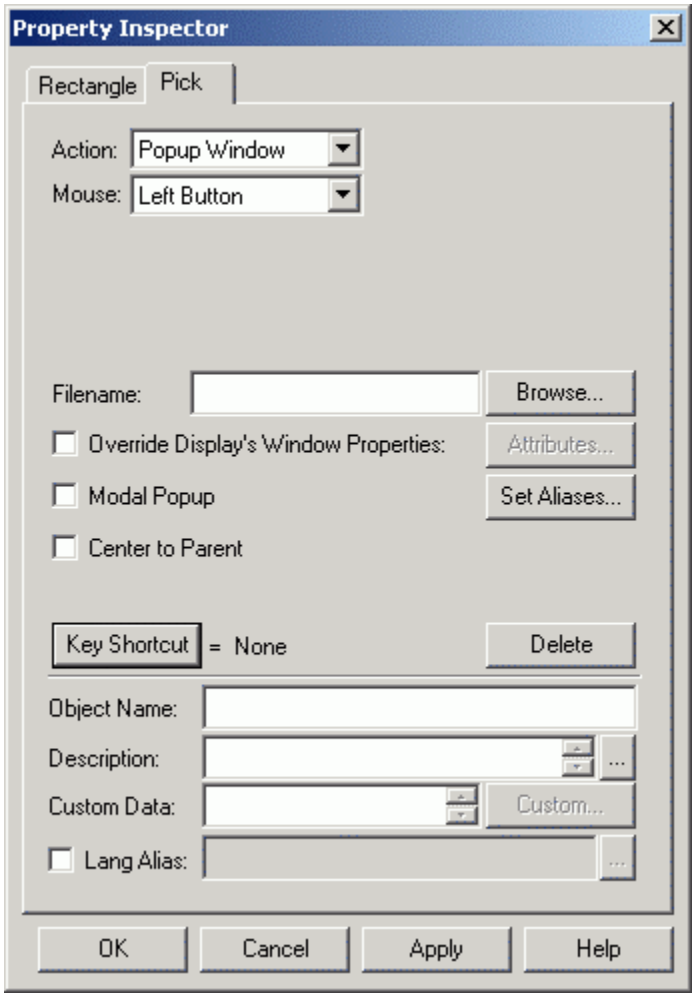
Note: The display file history (forward/back commands) remembers the initial alias settings specified when a display is opened and will reset those aliases when you go back to that display in the file history. Refer to the **Runtime Aliasing** section for more information.



Property Inspector Dialog Box: Display Forward Action

Action - Popup Window

The **Popup Window** pick action in the **Pick** tab of the Property Inspector opens a popup GraphWorX32 window in runtime mode.

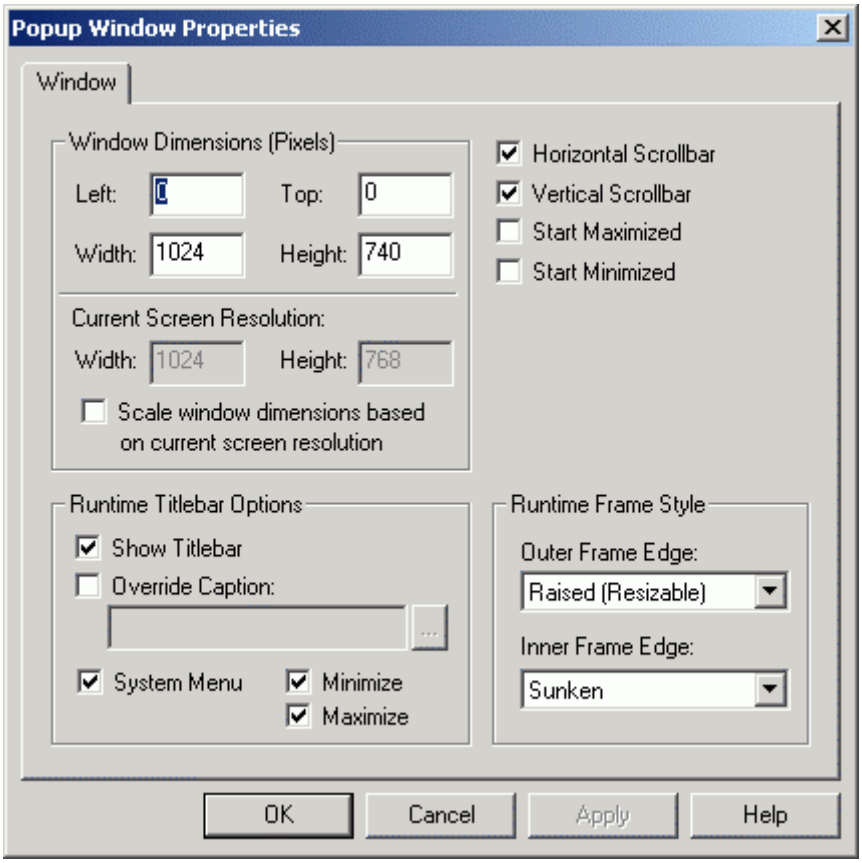


Property Inspector Dialog Box: Popup Window Action

Popup Window Parameters

Parameter	Description
File Name	Click the **Browse** button to pick a file name (the display file to be loaded into the popup window during runtime).
Override Display's Window Properties	Overrides the window properties of the display to be loaded.
Modal Popup	Opens the popup window as a modal window (i.e. disables the parent window).
Center to Parent	Automatically centers the popup window to the parent window when opened.

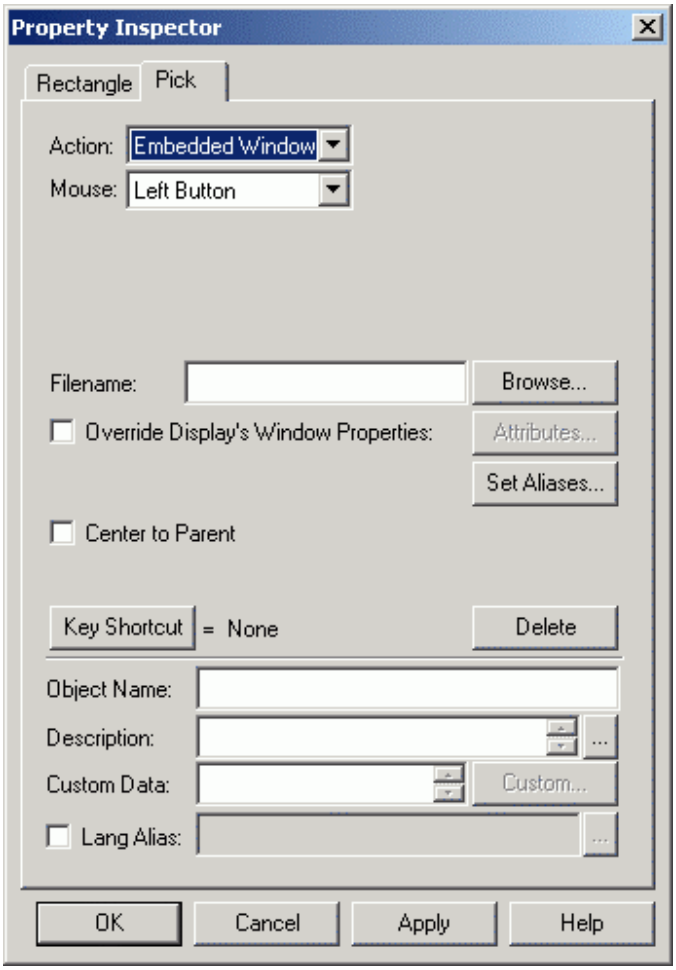
Attributes	Checking the **Override Display's Window Properties** check box enables the **Attributes** button. Clicking on this button opens the **Popup Window Properties** dialog box, which allows you to specify the window's dimensions and other attributes. Refer to the **Window** tab of the **Application Preferences** and **Display Properties** dialog boxes for more information.



Popup Window Properties

Action - Embedded Window

The **Embedded Window** pick action in the **Pick** tab of the Property Inspector is similar to the regular Popup Window pick action except that an embedded window is a "child" window of the display that launched the embedded window. The embedded window can move, scroll, and scale with the parent display. The parameter **Modal Popup** of the Popup Window action is not supported by the Embedded Window action.



Property Inspector Dialog Box: Embedded Window Action

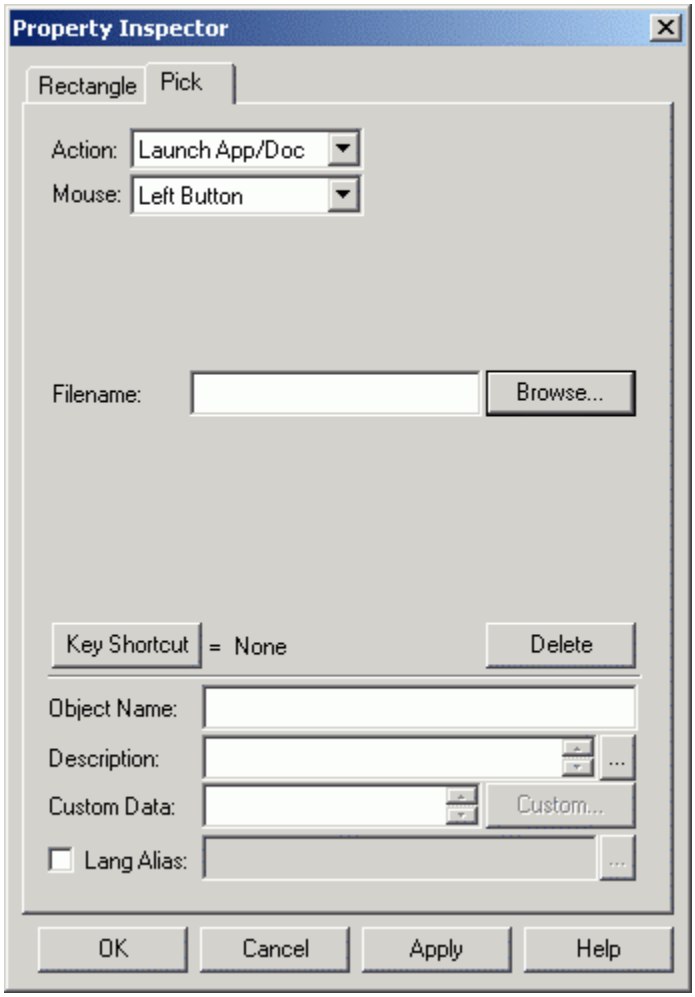
Embedded Window Parameters

Parameter	Description
File Name	Click the **Browse** button to pick a file name (the display file to be loaded into the popup window during runtime).
Override Display's Window Properties	Overrides the window properties of the display to be loaded.
Center to Parent	Automatically centers the popup window to the parent window when opened.
Attributes	Checking the **Override Display's Window Properties** check box enables the **Attributes** button. Clicking on this button opens the **Popup Window Properties** dialog box, which allows you to specify the window's dimensions and other attributes.

Action - Launch Application or Document

The **Launch Application/Document** pick action in the **Pick** tab of the Property Inspector launches the specified application or document when GraphWorX32 is in runtime mode. It provides the **Delete, Key Shortcut,** and **Mouse** parameters that have been discussed previously.

In addition to launching executables, you can launch various documents, including Microsoft Word documents, Microsoft Excel spreadsheets, HTML files, and text files. The registered application is automatically started when the file is launched.



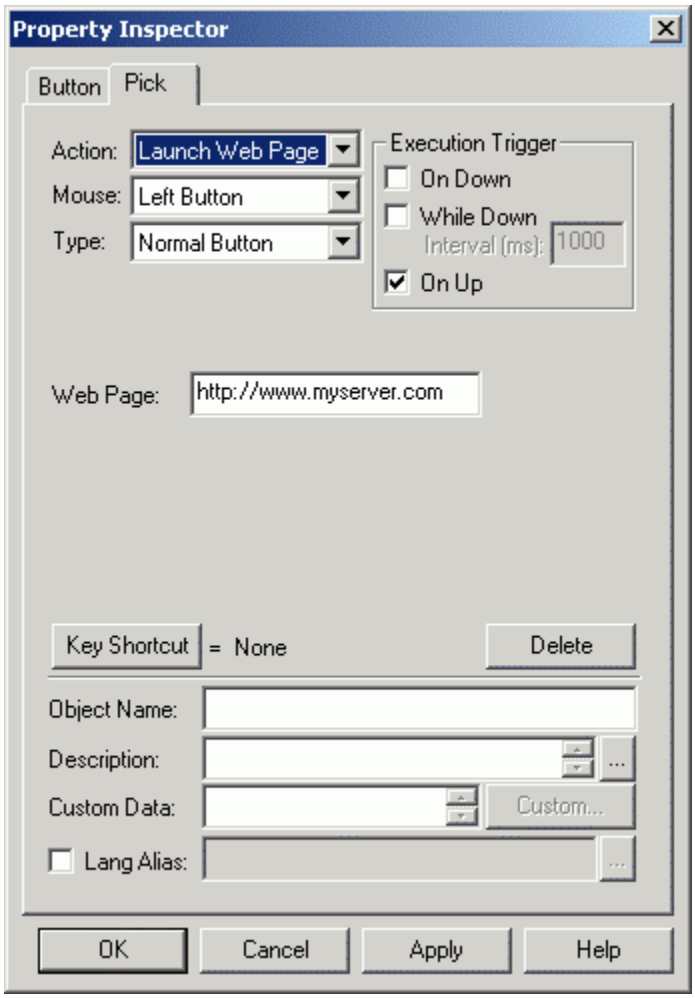
Property Inspector Dialog Box: Launch Application

Launch Application Parameters

Parameter	Description
File Name	Click the **Browse** button to pick an executable or document name (the file to be loaded during runtime).

Action - Launch Web Page

The **Launch Web Page** pick action in the **Pick** tab of the Property Inspector, shown in the figure below, launches a specified URL or Web site when GraphWorX32 is in runtime mode. Simply enter the URL address in the **Web Page** field, as shown in the figure below. When the action is executed in runtime mode, the specified Web page opens in the default Web browser.



Property Inspector Dialog Box: Launch Web Page Action

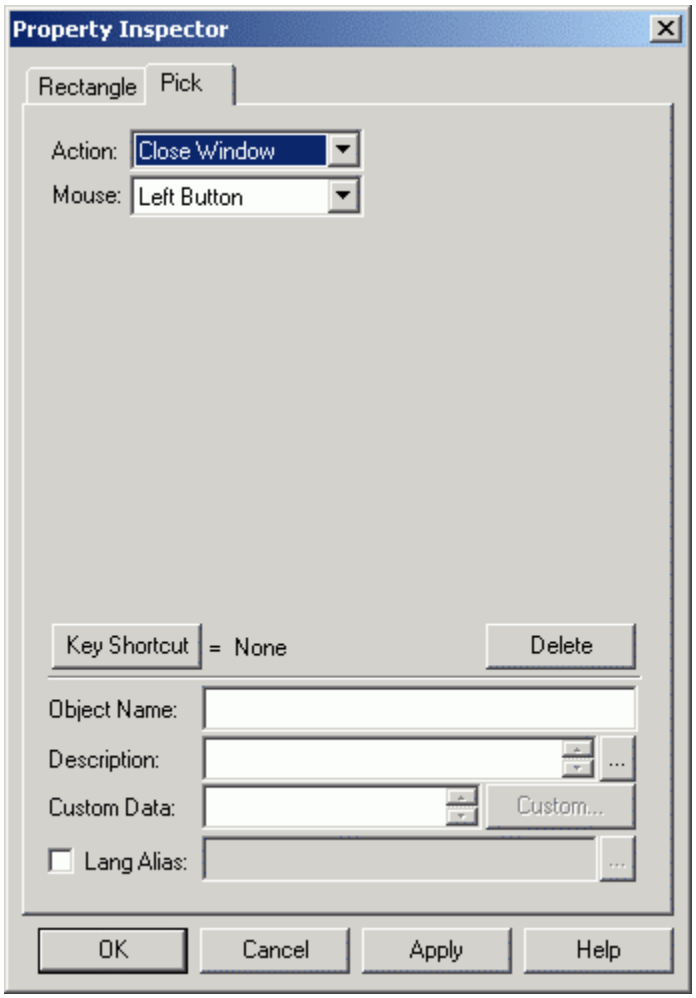
If you set this pick action for a push button, the button is automatically named "Go to" followed by the specified URL address, as shown in the figure below.



Push Button With Launch Web Page Pick Action

Action - Close Window

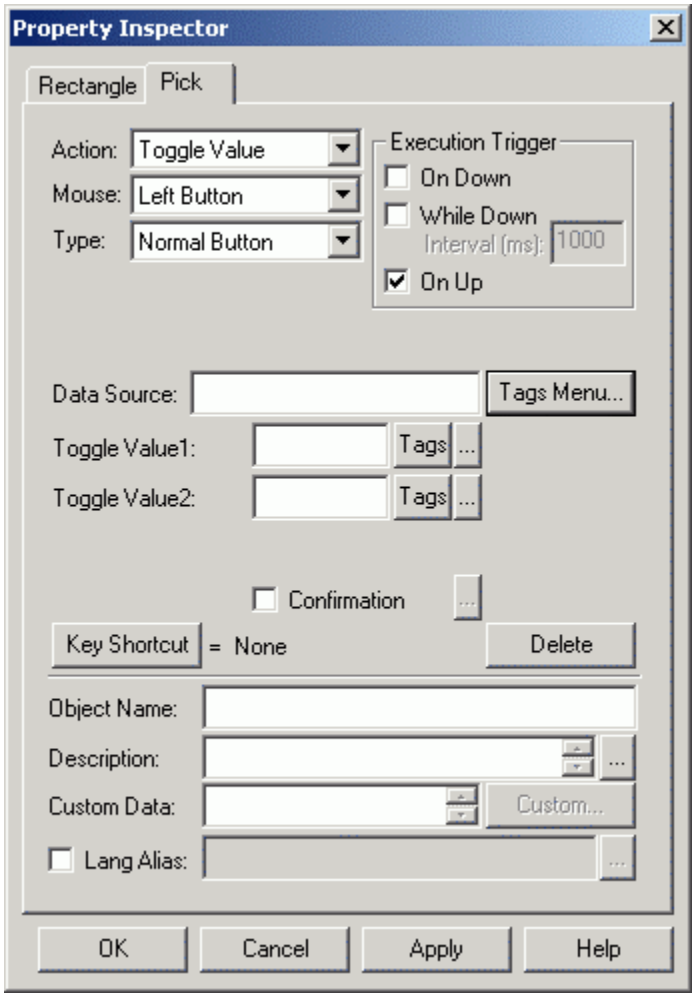
The **Close Window** pick action in the **Pick** tab of the Property Inspector closes the window to which this pick belongs. It provides the **Delete, Key Shortcut,** and **Mouse** parameters that have been discussed previously.



Property Inspector Dialog Box: Close Window Action

Action - Toggle Value

The **Toggle Value** pick action in the **Pick** tab of the Property Inspector is used to toggle between two specified values during runtime mode.



Property Inspector Dialog Box: Toggle Value Action

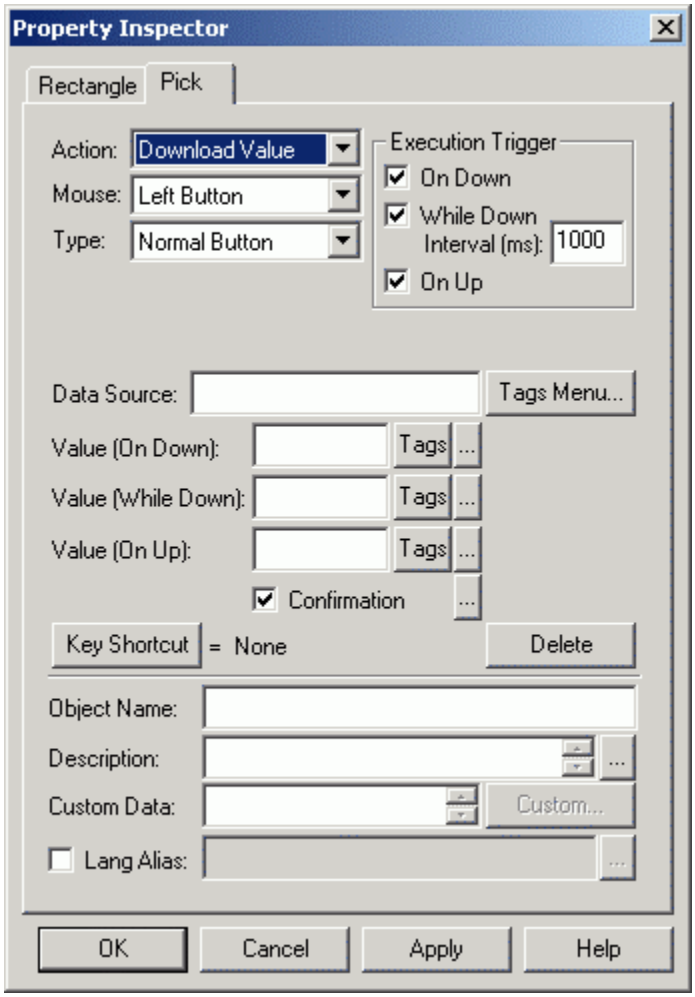
Toggle Value Parameters

Parameter	Description
Data Source	The data source is the tag to which the downloaded values are to be written. Click the **Tags Menu** button to quickly insert the desired OPC tag, alias, local variable, simulation variable, or global alias.
Toggle Value 1	Specifies the first toggle value when the button is clicked. The downloaded value can be a constant value, the value of a tag, the calculated value of an expression, an alias, or a global alias.
Toggle Value 2	Specifies the second toggle value when the button is clicked. The downloaded value can be a constant value,

	the value of a tag, the calculated value of an expression, an alias, or a global alias.
Confirmation	Defines a confirmation message, which is a message the GraphWorX32 prompts every time the user writes a value. Click the **…** button to open the **Confirmation Message Builder,** which helps you create confirmation messages. For more information, please see the Confirmation Message Builder section below.

Action - Download Value

The **Download Value** pick action in the **Pick** tab of the Property Inspector is used to download values when you click on the pick object.



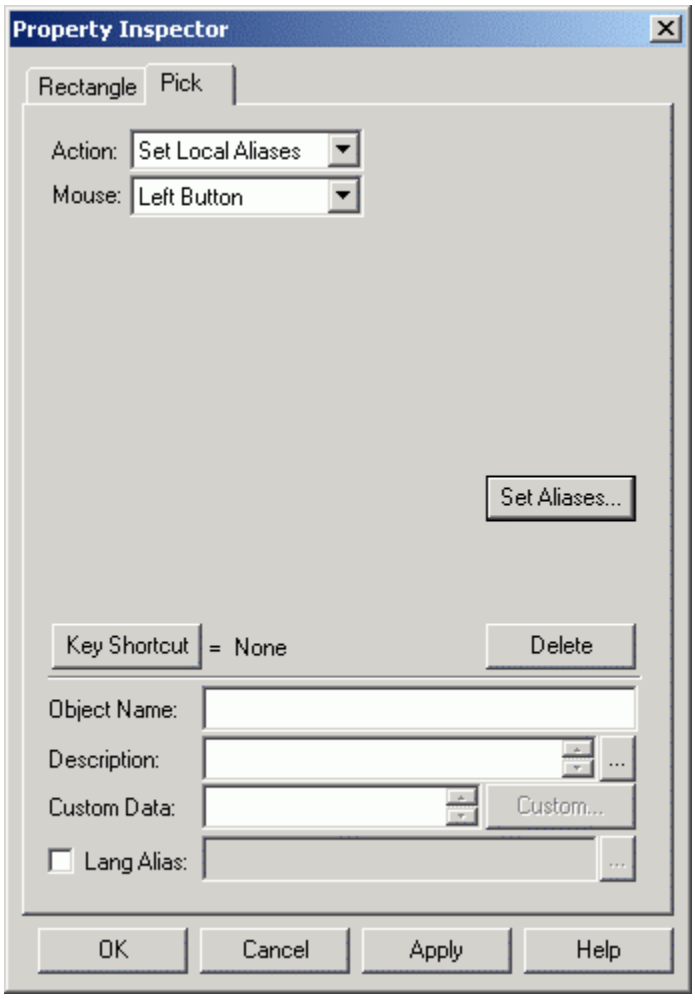
Property Inspector Dialog Box: Download Value Action

Download Value Parameters

Parameter	Description
Data Source	The data source is the tag to which the downloaded values are to be written. Click the **Tags Menu** button to quickly insert the desired OPC tag, alias, local variable, simulation variable, or global alias.
Values (On Down, While Down, On Up)	Specifies the values to be downloaded. The downloaded value can be a constant value, the value of a tag, the calculated value of an expression, an alias, or a global alias. **Note:** If no value is specified for the On-Up event, the pick action will display a number pad or a keypad dialog in runtime, where you can enter the value. This feature must be manually enabled in the **Runtime Advanced** tab of the **Application Preferences** dialog box by checking the **Ask for Download Value** check box.
Confirmation	Defines a confirmation message, which is a message the GraphWorX32 prompts every time the user writes a value. Click the **…** button to open the **Confirmation Message Builder,** which helps you create confirmation messages. For more information, please see the Confirmation Message Builder section below.

Action - Set Local Aliases

The **Set Local Aliases** pick action in the **Pick** tab of the Property Inspector sets aliases for data connections in runtime mode. Click the **Set Aliases** button on the property inspector to open the **Set Aliases Configuration** dialog box. Refer to the section **Aliasing Data Connections** for details.

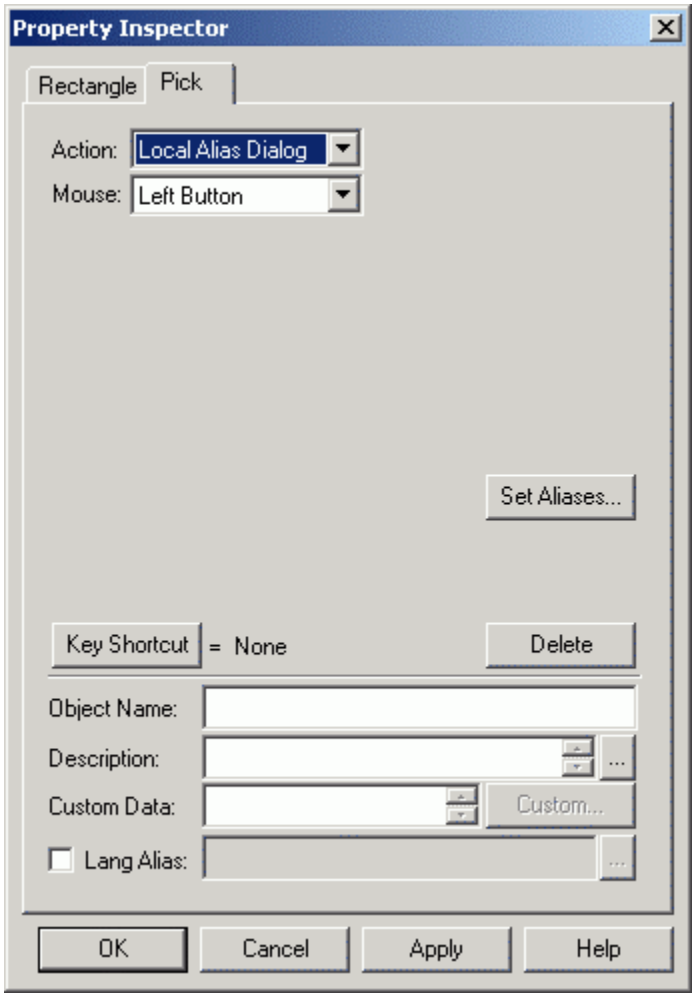


Property Inspector Dialog Box: Set Local Aliases Action

Action - Local Alias Dialog

This **Local Alias Dialog** pick action in the **Pick** tab of the Property Inspector displays the Aliases dialog for the display objects or for the whole display in runtime mode. Click the **Set Aliases** button on the Property Inspector to open the **Set Aliases Configuration** dialog box.

Refer to the section **Aliasing Data Connections** for details.

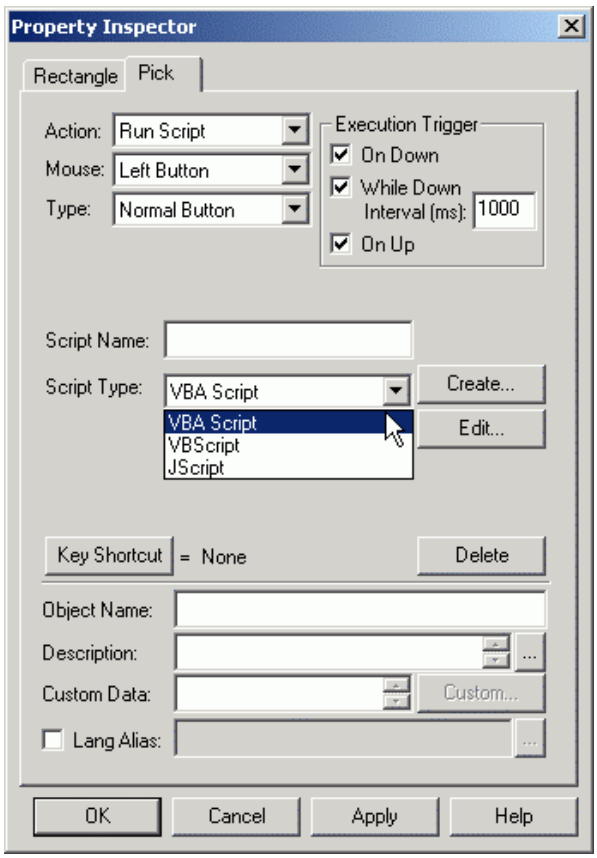


Property Inspector Dialog Box: Local Alias Dialog Action

Action - Run Script

The **Run Script** pick action in the **Pick** tab of the Property Inspector runs three different types of scripts: VBA, VBScript and JScript.

1. Select the type from the **Script Type** drop-down list, as shown in the figure below.



Choosing a Script Type for the Run Script Pick Action

2. Once you have chosen a language, the corresponding editor can be used to write the code for the pick action. When you choose VBScript or JScript, a new script is generated in the **Script Editor.**

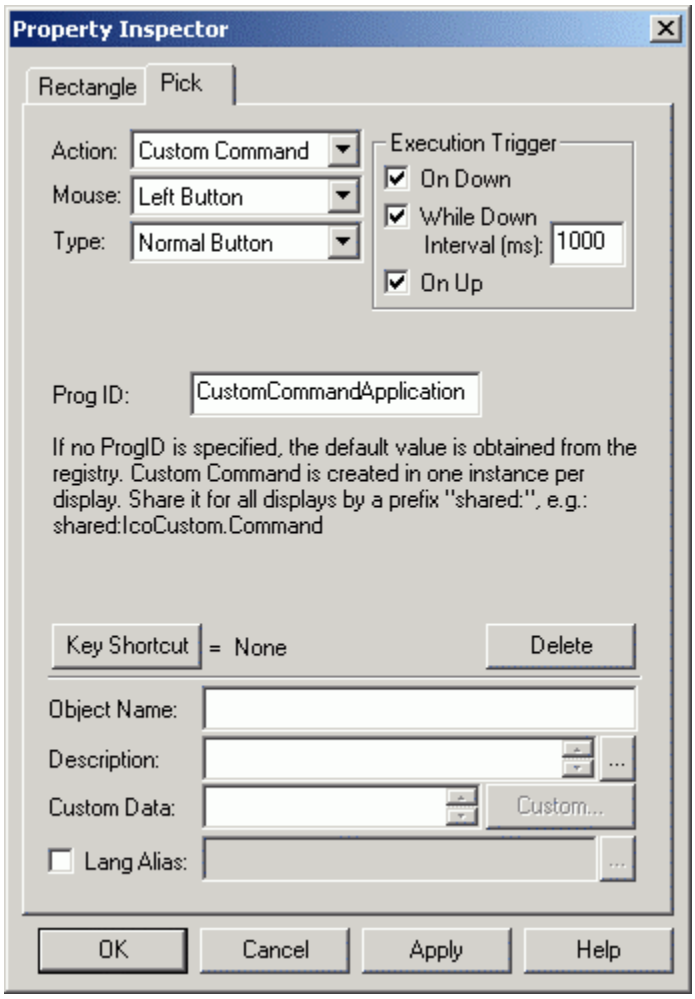
Run Script Parameters

Parameter	Description
Script Name	The name of the VBA macro to be executed. This name is saved in the GraphWorX32 file.
Script Type	Select VBA Script, VBScript or Jscript.
Create	Opens the **VBA Script Wizard,** which allows you to create a macro name.
Edit	Opens the corresponding script editor, which can be used to write the code for the pick action.

Action - Custom Command

The **Custom Command** pick action in the **Pick** tab of the Property Inspector can be used to run a custom function. This function can be an executable file (.exe) or a .dll file. Refer to the **Custom Configuration** section for details.

GraphWorX32 supports multiply custom command DLLs. In the **Prog ID** field, you can specify the Prog ID of your custom command component. If you do not provide a value, the Prog ID will be obtained from the registry by default as "IcoCustomCommand."



Property Inspector Dialog Box: Custom Command Action

You can run custom functions / applications using the Custom Command option pick action. The custom application has to be specified by the user. The application /function is either an executable (.EXE) or a DLL.

The first time the user clicks on a pick action that is a **Custom Command,** GraphWorX32 creates an Automation object that has a certain program ID (ProgID).

By default the ProgID is: **IcoCustom.Command**. This ProgID can be overridden by setting an alternative ProgID in the registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\Gwx32\OEM Information\Custom Command ProgID = "YourProgIDHere"

You will probably want this Automation object to be exposed from an EXE. The EXE should allow you to create multiple instances of this automation object without launching multiple instances of the application.

GraphWorX32 then uses the dispatch interface of this object to call the method:

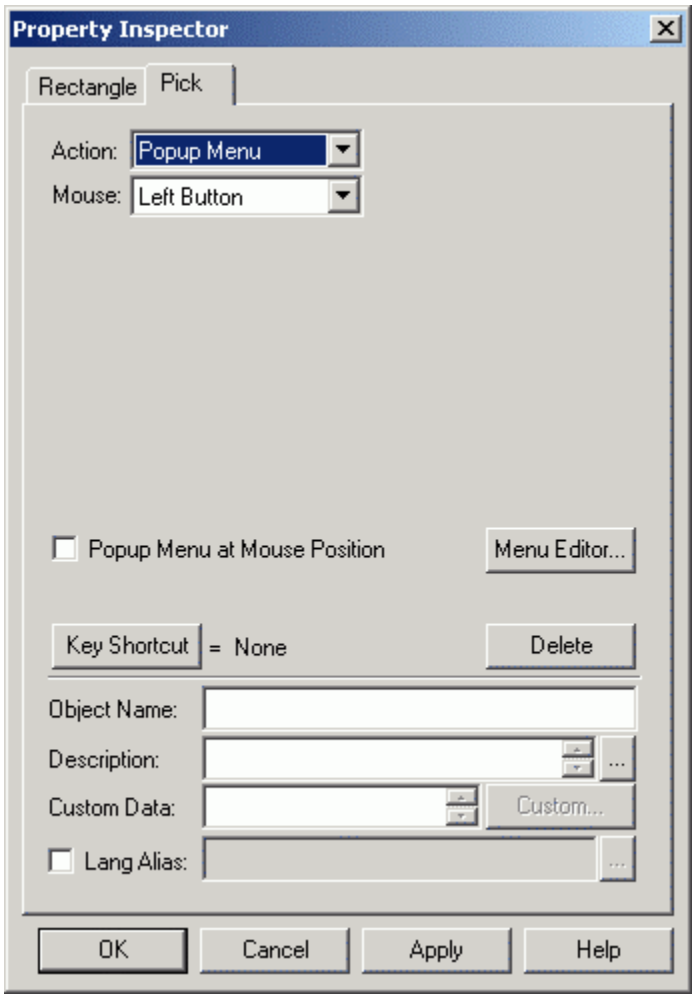
boolean **CustomCommand(**long hWndParent, IDispatch* ObjectOne, BSTR ObjectOneType, IDispatch* ObjectTwo, BSTR ObjectTwoType**)**;

The parameters are the same as CustomConfigure() described in the previous section. DO NOT Release() the dispatch pointers passed to this method!!!

After the method returns, GraphWorX32 does NOT release the dispatch pointer for the "IcoCustom.Command" automation object. GraphWorX32 keeps the dispatch pointer for the next time a Custom Command is executed (so that GraphWorX32 does not have to create the object again (so execution is faster). This dispatch pointer is not released until this instance of GraphWorX32 is closed.

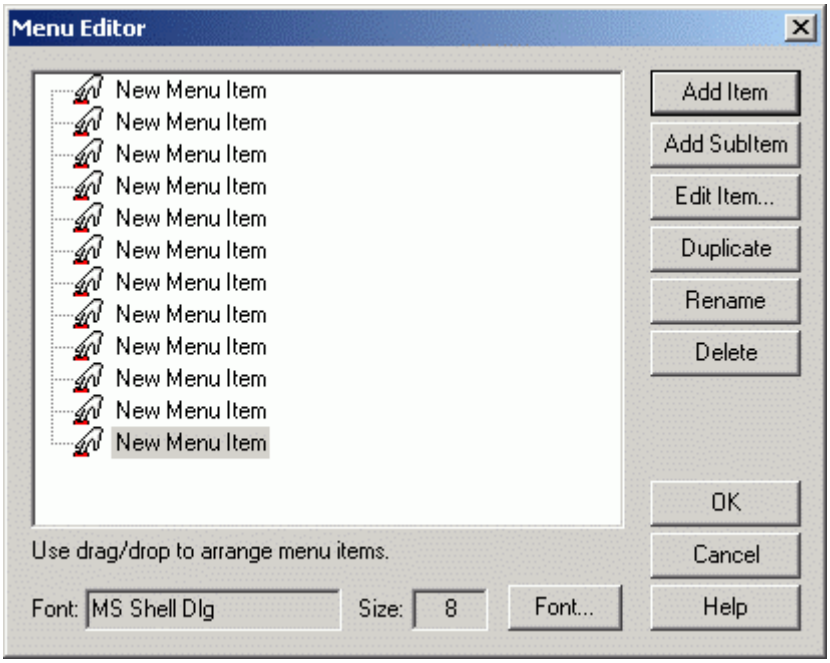
Action - Popup Menu

The **Popup Menu** pick action in the **Pick** tab of the Property Inspector configures the font and size for popup menus opened from a pick action. You can also configure the **Popup Menu at Mouse Position.**



Property Inspector Dialog Box: Popup Menu Action

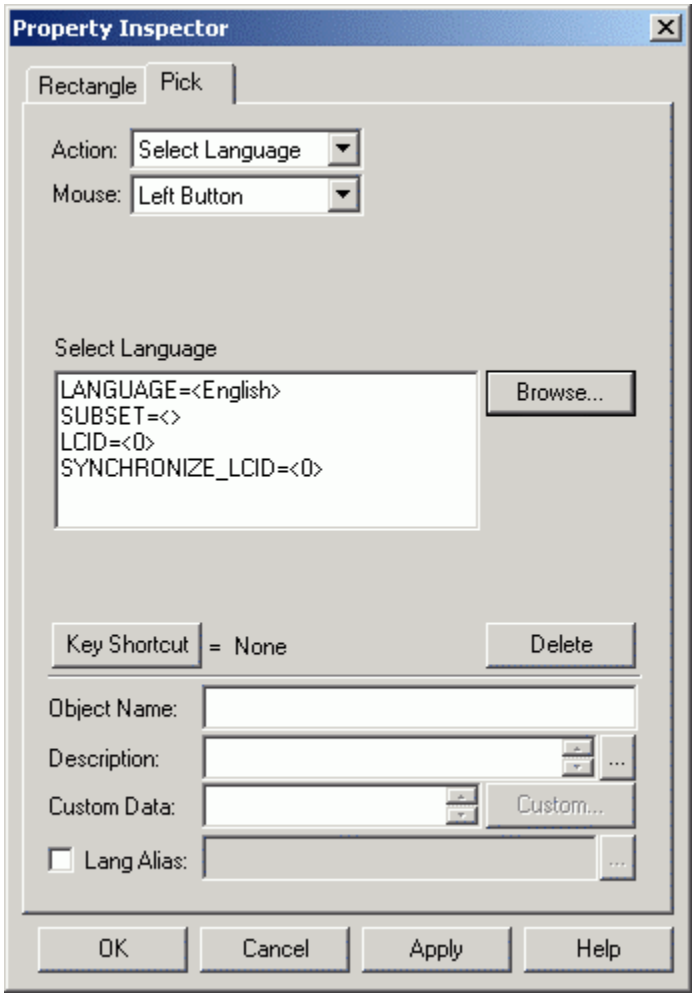
Click **Menu Editor** to open the **Menu Editor** dialog box, shown below, which includes fields for **Font** and **Size.** Click the **Font** button to select a font. In addition, the font face can be changed based on the loaded language by the language alias placed on the **Pick** tab. During runtime mode, the font is picked up from the language database based on the language alias and is assigned to the popup menu. This can be useful if the current/default font does not support all characters of a given language.



Menu Editor Dialog Box

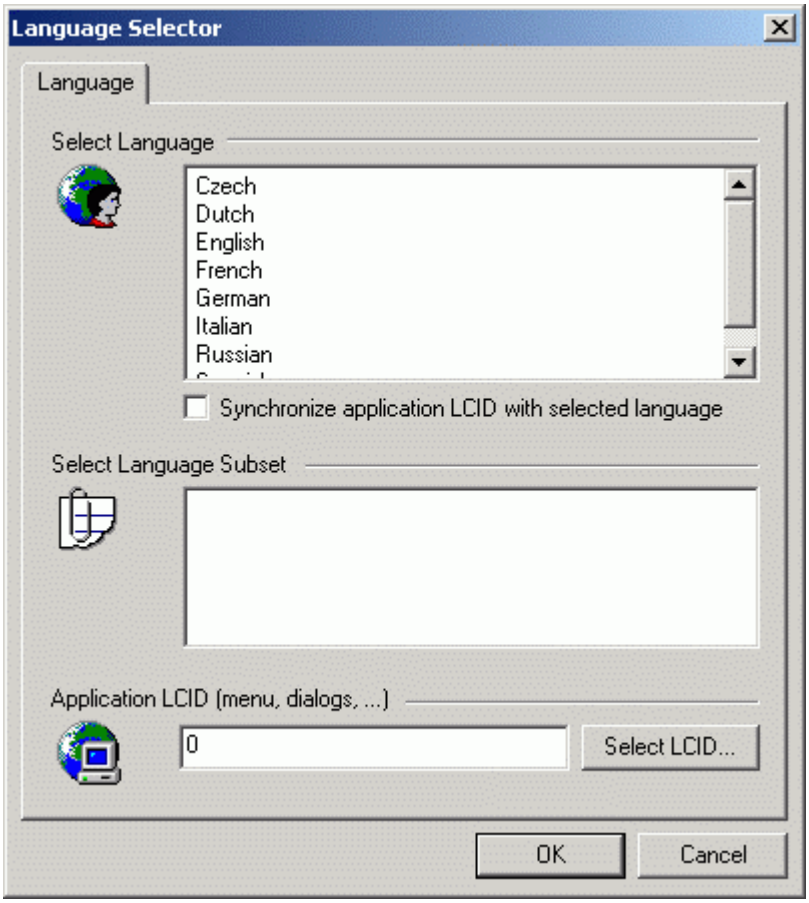
Action - Select Language

The **Select Language** pick action in the **Pick** tab of the Property Inspector specifies languages from a pick action.



Property Inspector Dialog Box: Select Language Action

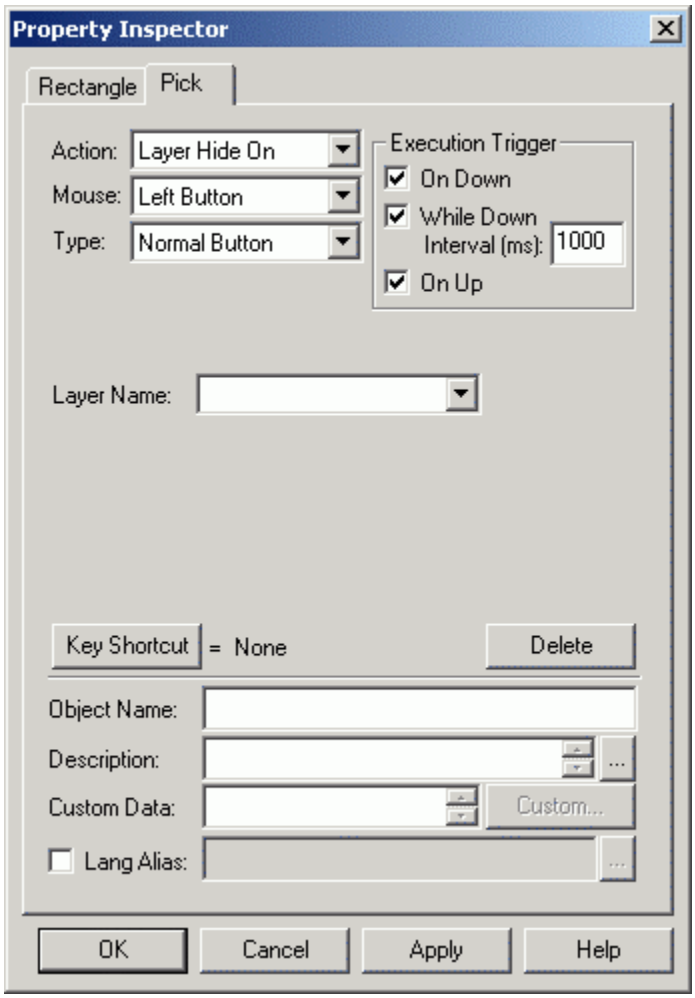
Clicking the **Browse** button opens the **Language Selector** dialog box, shown below, which enables you to choose from a list of languages in the language-switching database.



Selecting a Language

Action - Layers

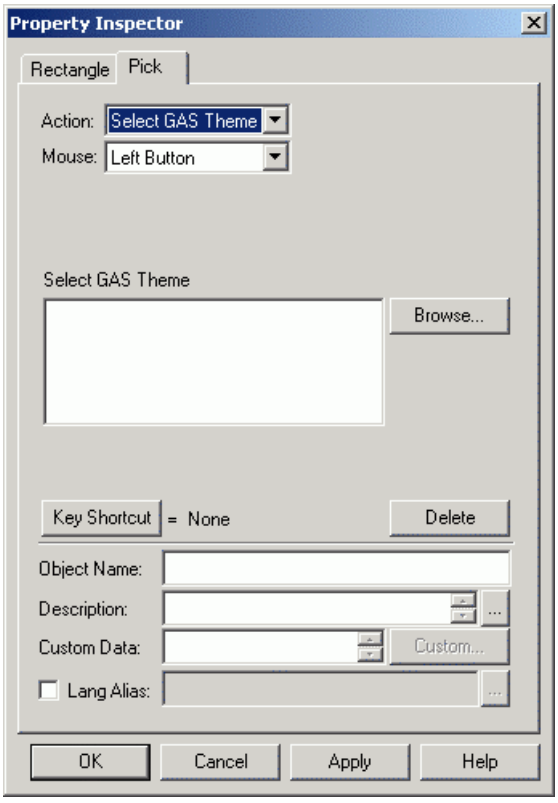
You can hide, show, and toggle layers using the pick actions **Layer Hide On, Layer Hide Off,** and **Layer Hide Toggle** in the **Pick** tab of the **Property Inspector** dialog box, as shown in the figure below. Select the appropriate pick action, and then select a layer from the **Layer Name** drop-down list, as shown in the figure below.



Property Inspector Dialog Box: Layers Actions

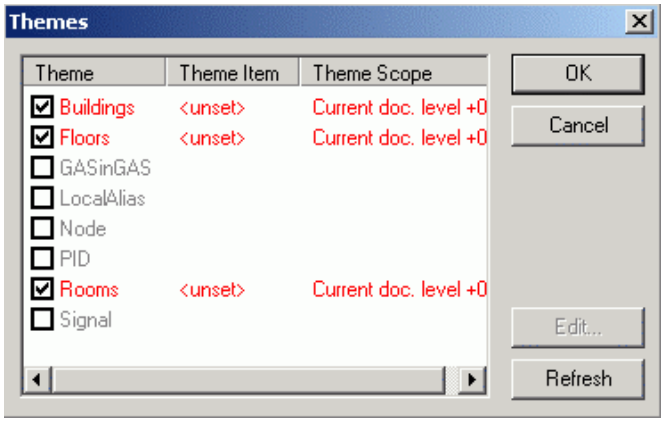
Action - Select Global Alias Theme

The **Select GAS Theme** pick action in the **Pick** tab of the Property Inspector allows you to select global alias themes, as shown in the figure below. Click the **Browse** button to select a theme from the **Themes** dialog box, as shown in the figure below. Click **OK.**



Property Inspector Dialog Box: Select Global Alias Theme Action

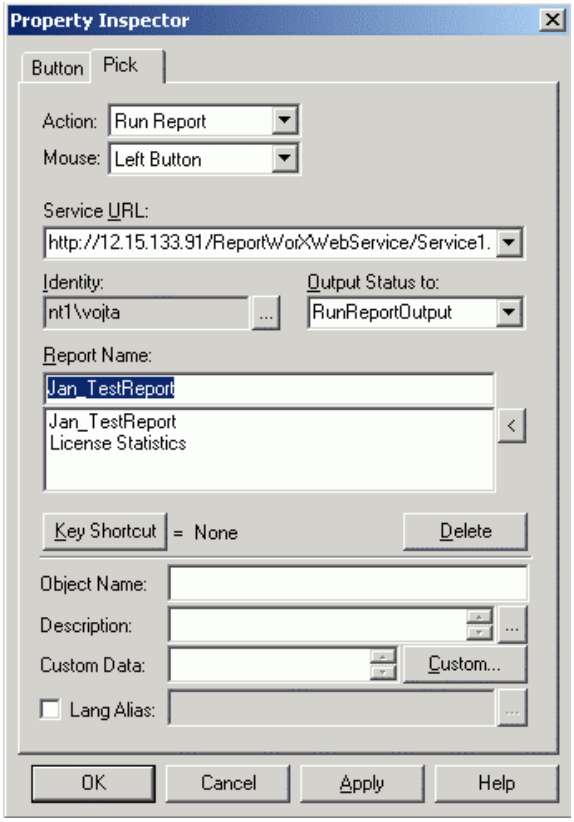
Note: To edit a theme's properties, select a theme and click the **Edit** button. This opens the **Theme Editor,** where you can specify theme items and the theme scope. For information about editing global alias themes and specifying the theme scope, please see the **Runtime Advanced Settings** section above.



Choosing Global Alias Themes

Action - Run Report

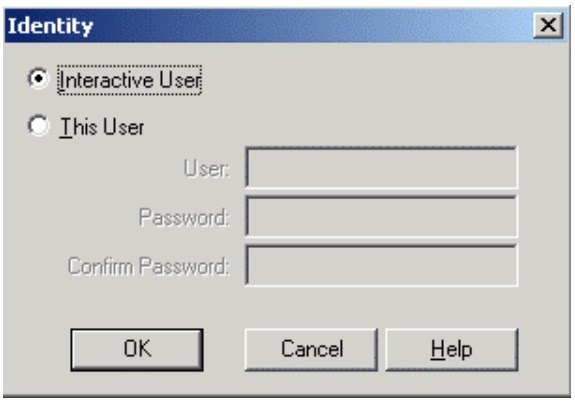
The **Run Report** pick action in the **Pick** tab of the Property Inspector allows you to execute a ReportWorX report during runtime. For more information about ReportWorX, please refer to the ReportWorX documentation on the ICONICS BizViz product CD.



Property Inspector Dialog Box: Run Report Action

Run Report Parameters

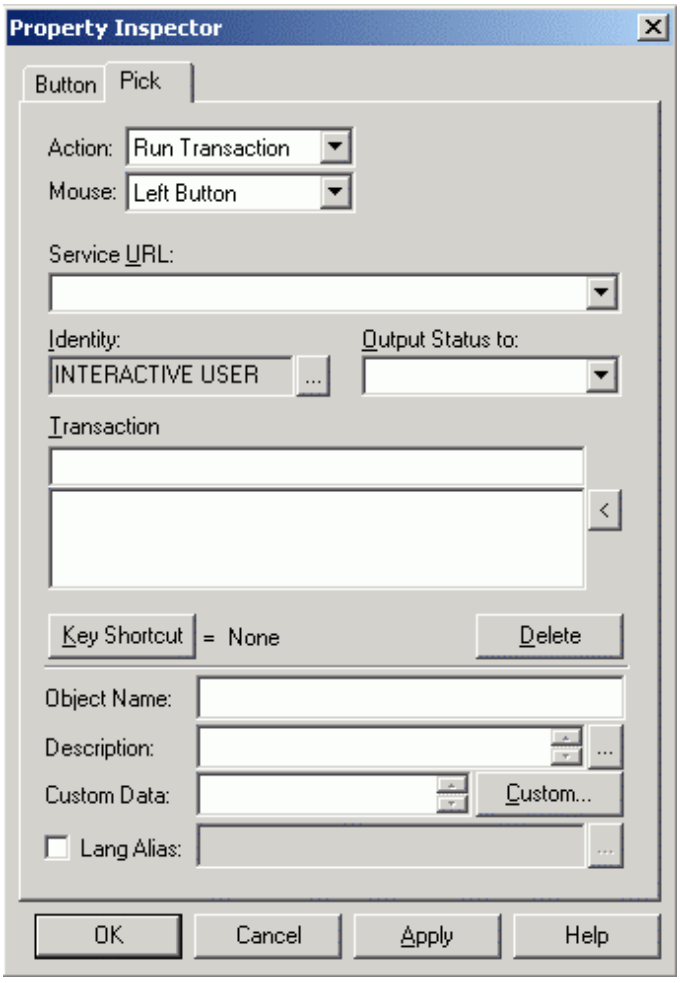
Parameter	Description
Service URL	Specifies the URL address of the ReportWorX server.
Identity	If a user name and password are required to access the ReportWorX server, click the **…** button next to the Identity field. This opens the **Identity** dialog box, shown in the figure below, where you can specify the user name and password.
Output Status To	From the drop-down list, select one of the text objects, which already have an object name assigned. If selected, the output is sent to this text object.
Report Name	Select from the list the name of the report that should be activated. Refresh the list by the clicking the **<** button to the right.



Report User Identity

Action - Run Transaction

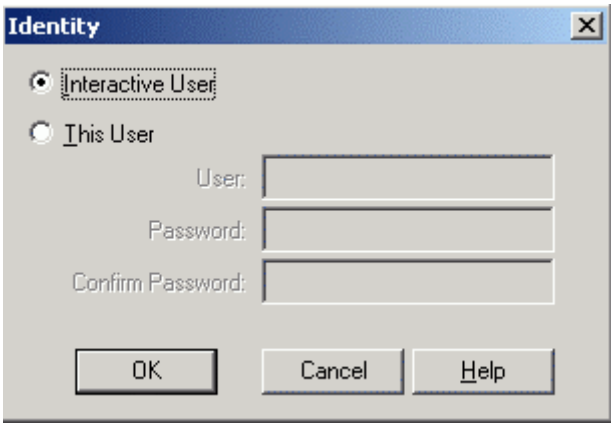
The **Run Transaction** pick action in the **Pick** tab of the Property Inspector allows you to execute a BridgeWorX transaction during runtime. For more information about BridgeWorX, please refer to the BridgeWorX documentation on the ICONICS BizViz product CD.



Property Inspector Dialog Box: Run Transaction Action

Run Transaction Parameters

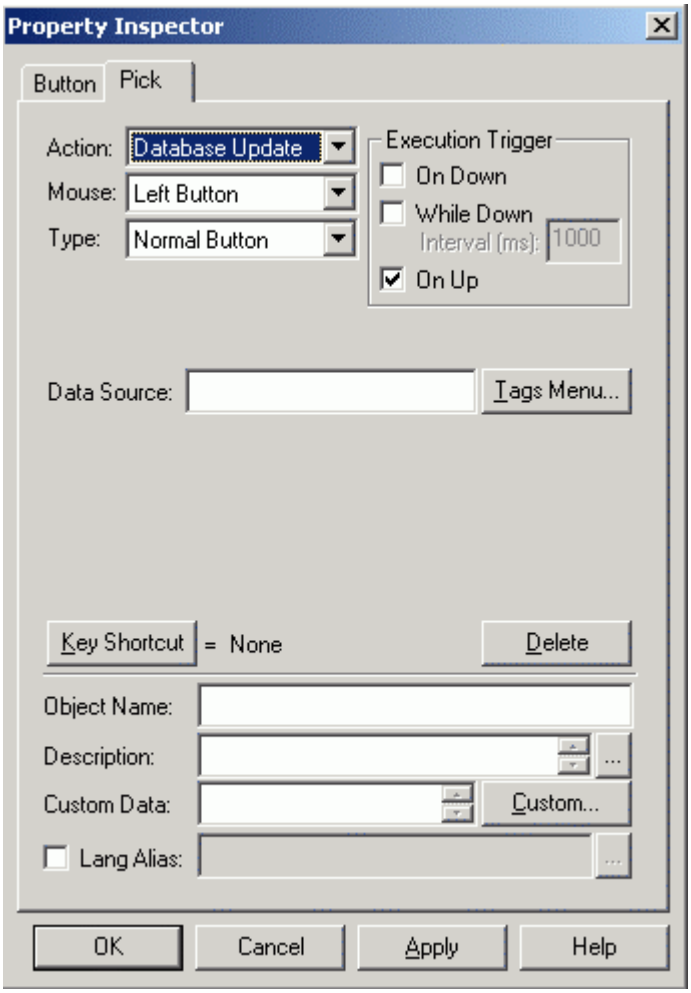
Parameter	Description
Service URL	Specifies the URL address of the BridgeWorX server.
Identity	If a user name and password are required to access the BridgeWorX server, click the **…** button next to the Identity field. This opens the **Identity** dialog box, shown in the figure below, where you can specify the user name and password.
Output Status To	From the drop-down list, select one of the text objects, which already have an object name assigned. If selected, the output is sent to this text object.
Transaction Name	Select from the list the name of the transaction that should be activated. Refresh the list by the clicking the **<** button to the right.



Transaction User Identity

Action - Database Update

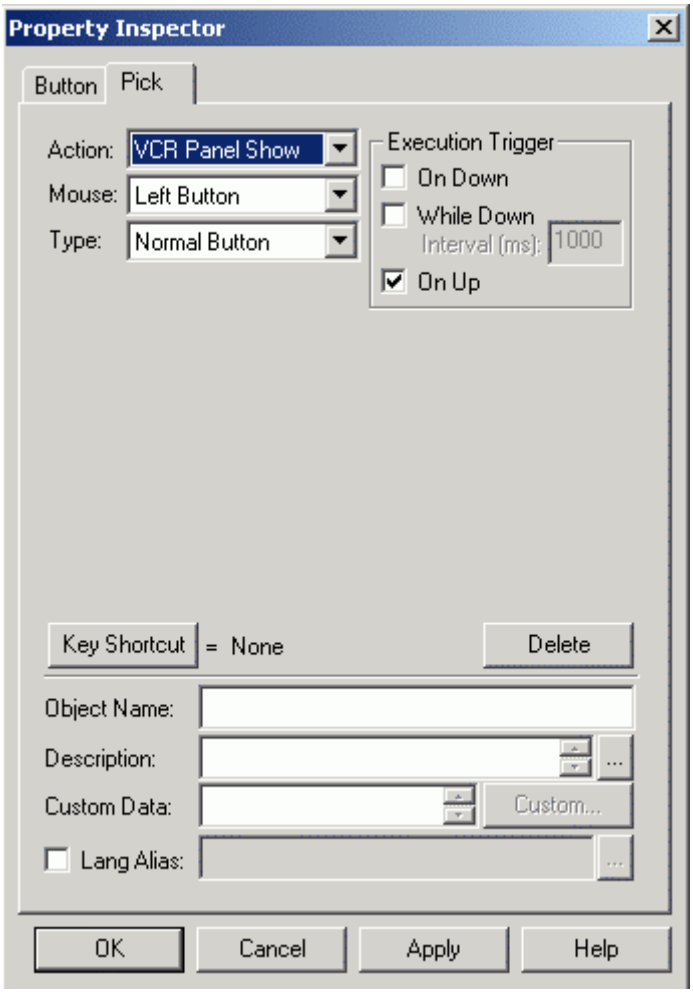
The **Database Update** pick action in the **Pick** tab of the Property Inspector updates the data mining server by writing any value to a data mining tag exposed by this server. To specify a data tag, click on the Tags Menu button and select a data mining tag from the **Database Access** tab of the **Unified Data Browser.** For more information, please see the Data Mining Configurator documentation.



Property Inspector Dialog Box: Database Update Action

Action - VCR Panel Show/Hide

The **VCRPanel Show/Hide** pick actions in the **Pick** tab of the Property Inspector simply display or hide the VCRWorX32 Data Replay control panel during runtime mode. Click **OK.** For more information about VCRWorX, please see the VCRWorX help documentation.

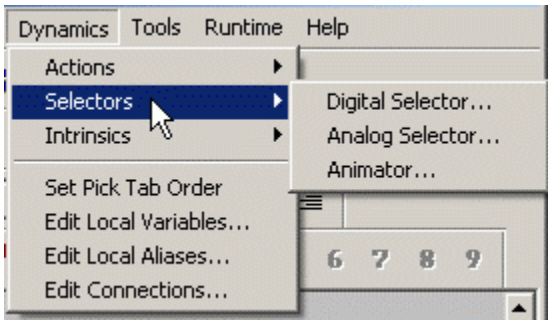


Property Inspector Dialog Box: VCR Panel Show Action

Selector Dynamics

Selectors display specific objects based on the value of a data connection. You can access selectors by choosing **Selectors** from the **Dynamics** menu, as shown above, or from the **Dynamics** toolbar. There are three types of selectors:

- **Digital Selector.** Displays one of a number of objects based on the state of digital signals. One signal is connected to each object. The highest priority signal (determined by position in the list) displays the corresponding object when true (or optionally false).
- **Analog Selector.** Displays one of a number of objects based on the value of the analog signal. This object displays when the signal falls within the range of values specified for the object.
- **Animator.** Selects a group of objects to display sequentially based on the state of a digital variable to which they are connected.



Selector Menu

Note: The digital selector, analog selector, and animator can be modified after creation by using subsymbol editing. Choose the selector and either do **SHIFT+Right-click** or **Right-click - Edit Symbol** to enter subsymbol editing. Now you can add, modify, and delete the objects in the digital selector. Press the **Escape** key or double-click outside the object to close the subsymbol editing. The changes are reflected in the selector. Use the Property Inspector to connect newly added objects if necessary. An alternative way is to use the **Edit > Find** and **View** properties, or to select the inner objects from the tree.

Digital Selector



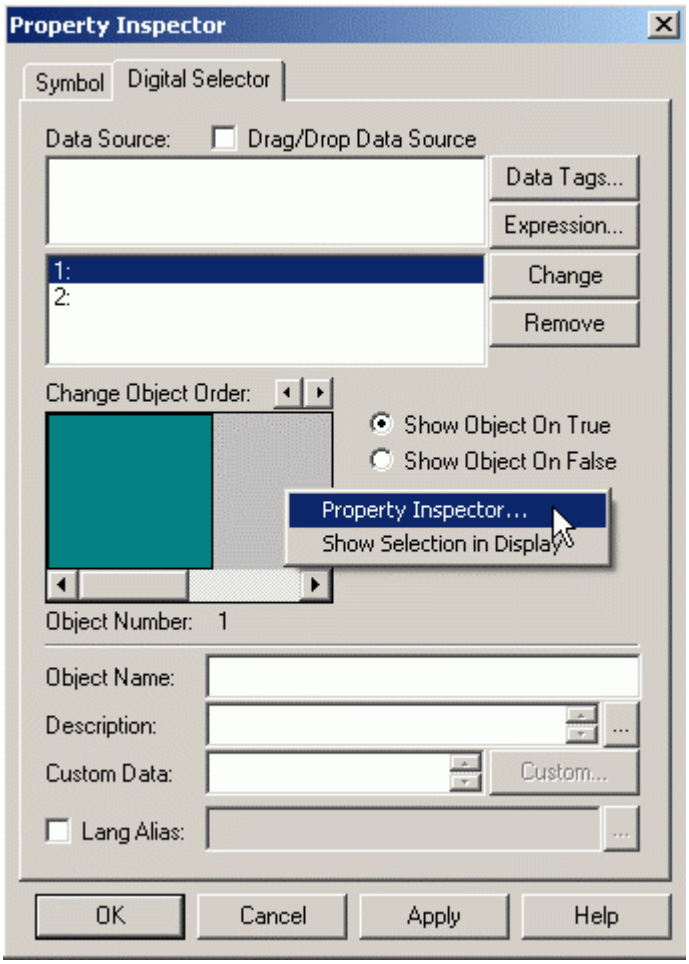
The **Digital Selector** enables you to connect individual objects to corresponding digital data points. During runtime mode, when the connected data point goes true (logical 1)(or optionally false), the connected object appears on the screen.

Only one object per connected data point can be displayed at a time. For example, if two objects are connected to the same digital data point, the object closer to the top of the connection list first appears. Generally, no objects display when all signals are false (logical 0).

To define a digital selector:

1. Select two or more objects.

2. Choose **Selectors > Digital Selector** from the **Dynamics** menu, or click the **Digital Selector** button on the **Dynamics** toolbar. The **Digital Selector** tab of the Property Inspector opens, as shown below.
3. Select the desired parameters listed in the table below.
4. Establish a data-point connection for each object using the same procedures, and then click **OK.** The objects to which the digital selector is connected are grouped together in the display.



Property Inspector Dialog Box: Digital Selector Tab

Digital Selector Parameters

Parameter	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.
Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Change Button	Changes existing data connections. The currently selected item in the Data Source list box is changed to the string currently in the data source edit control.
Remove Button	Removes currently selected data connections from the data source list box.
Object Numbers	Lists the objects in the order created and their data-point connections assigning priorities to the objects.
Change Object Order	Clicking the left and right arrows to change the object order assigns new priorities for the data connections. Double-clicking one of the child objects in the **Change Object Order** (preview) window opens the Property Inspector for the selected child object. Right-clicking a child object (e.g. ellipse) in the **Change Order** (preview) window and selecting **Property Inspector** opens the Property Inspector for the selected child object. Right-clicking a child object (e.g. ellipse) in the **Change Object Order** (preview) window and choosing **Show Selection in Display** selects the current child object in the main display, as shown in the figure below. The Property Inspector can be closed and the child (which is still selected) can be modified or deleted. This function uses the sub-symbol editing feature and opens the Selector/Animator symbol for editing. It is possible to add, edit or delete all child objects within the symbol. To close the sub-symbol editing, press the **Esc** key or double-click outside the bounding rectangle.
Show Object	Shows the object when the connected data point goes

on True	true (logical 1).
Show Object on False	Shows the object when the connected data point goes false (logical 0).
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

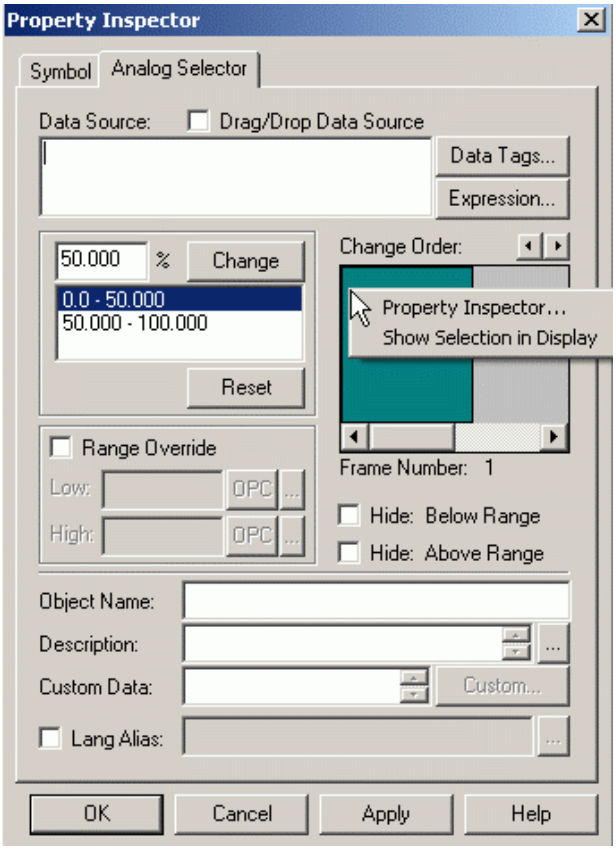
Analog Selector



The **Analog Selector** dynamic defines a group of objects attached to an analog data point. During runtime mode, the value of the analog data point determines which assigned object is shown.

To define an analog selector:

1. Select two or more objects in the display.
2. Choose **Selectors > Analog Selector** from the **Dynamics** menu, or click the **Analog Selector** button on the **Dynamics** toolbar. This opens the **Analog Selector** tab on the Property Inspector, as shown below.
3. Select the parameters listed in the table below. Click **OK** when you are finished. The objects to which the analog selector is connected are grouped together in the display.



Property Inspector Dialog Box: Analog Selector Tab

Analog Selector Parameters

Parameter	Description
Data source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.
Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Percentage	Allows objects to display when the data-point value is at a certain percentage of the full range.

Percentage List	Displays the percentage for each object.
Change Button	Changes the percentage chosen if you type a new value in the percentage field.
Reset Button	Resets the percentages chosen for the items to the default values. (The default values are an even distribution within the available range based on the number of objects selected.)
Change Order	Clicking the left and right arrows to change the object order assigns new priorities for the data connections. Double-clicking one of the child objects in the **Change Object Order** (preview) window opens the Property Inspector for the selected child object. Right-clicking a child object (e.g. ellipse) in the **Change Order** (preview) window and selecting **Property Inspector** opens the Property Inspector for the selected child object. Right-clicking a child object (e.g. ellipse) in the **Change Object Order** (preview) window and choosing **Show Selection in Display** selects the current child object in the main display, as shown in the figure below. The Property Inspector can be closed and the child (which is still selected) can be modified or deleted. This function uses the sub-symbol editing feature and opens the Selector/Animator symbol for editing. It is possible to add, edit or delete all child objects within the symbol. To close the sub-symbol editing, press the **Esc** key or double-click outside the bounding rectangle.
Range Override	Activates an operating range other than the default range for the data point. Specifies both a High Range and a Low Range.
Hide Below Range	Hides all the associated objects when the value of the connected data point is below the specified range.
Hide Above Range	Hides all the associated objects when the value of the connected data point is above the specified range.
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.

Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

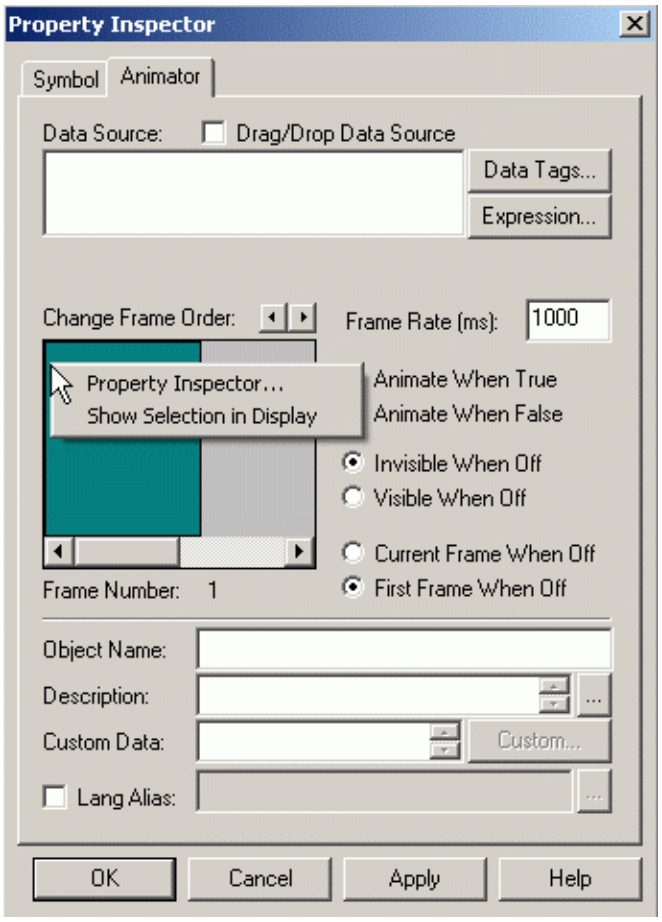
Animator



The **Animator** connection enables you to select a group of objects to display sequentially based on the state of a digital variable to which they are connected. When that digital variable goes to its true state (logical 1) (or optionally false) during runtime mode, the assigned objects appear on screen in the sequence in which they are ordered. The sequence repeats until the connected process variable returns to a false state (logical 0, or optionally true).

To make an animator connection:

1. Select two or more objects.
2. Choose **Selectors > Animator** from the **Dynamics** menu, or click the **Animator** button on the **Dynamics** toolbar. The Animator tab on the Property Inspector opens, as shown below.
3. Select the parameters listed in the table below, and then click **OK.**



Property Inspector Dialog Box: Animator Tab

Animator Parameters

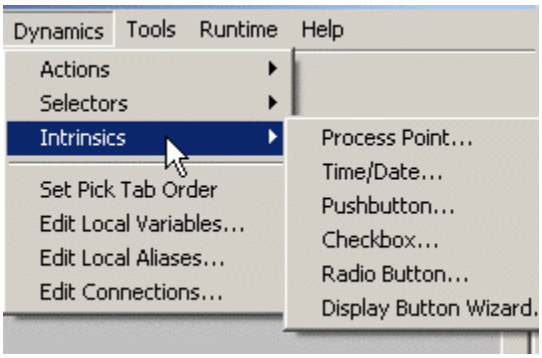
Parameters	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.

Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Change Frame Order	Clicking the left and right arrows to change the object order assigns new priorities for the data connections. Double-clicking one of the child objects in the **Change Object Order** (preview) window opens the Property Inspector for the selected child object. Right-clicking a child object (e.g. ellipse) in the **Change Order** (preview) window and selecting **Property Inspector** opens the Property Inspector for the selected child object. Right-clicking a child object (e.g. ellipse) in the **Change Object Order** (preview) window and choosing **Show Selection in Display** selects the current child object in the main display, as shown in the figure below. The Property Inspector can be closed and the child (which is still selected) can be modified or deleted. This function uses the sub-symbol editing feature and opens the Selector/Animator symbol for editing. It is possible to add, edit or delete all child objects within the symbol. To close the sub-symbol editing, press the **Esc** key or double-click outside the bounding rectangle.
Frame Number	Displays the sequence number of the currently displayed object.
Frame Rate	The rate (in milliseconds) at which the frames will change while animating.
Animate When True	Animates the object when the digital variable goes to its true state (logical 1).
Animate When False	Animates the object when the digital variable goes to its false state (logical 0).
Invisible When Off	Displays no objects when not animating.
Visible When Off	Displays the specified object when not animating.
Current Frame When Off	Displays the most recent animation frame when not animating.
First Frame When Off	Displays the first frame of the animation when not animating.
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips.

	what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

Intrinsic Dynamics

The **Intrinsic** dynamics enable you to create operator controls in display files. You can make dynamic connections to control objects that display real-time data during runtime mode, such as data entry objects, sliders, and push buttons. You can access the control functions from the **Intrinsics** submenu of the **Dynamics** menu, as shown below, or from the **Dynamics** toolbar.



Dynamics - Intrinsics Menu

Process Points and Data Entries
State Fields
Time/Date
Push Button
Check Box
Radio Button
Display Button Wizard

Process Points and Data Entries



The Process Point/Data Entry (PPT/DE) function creates an object used to display and enter data in an alphanumeric format. A **process point** is an object that displays the real-time value of a connected data point during runtime mode. This value is updated when the server notifies GraphWorX32 of a change in the data-point value.

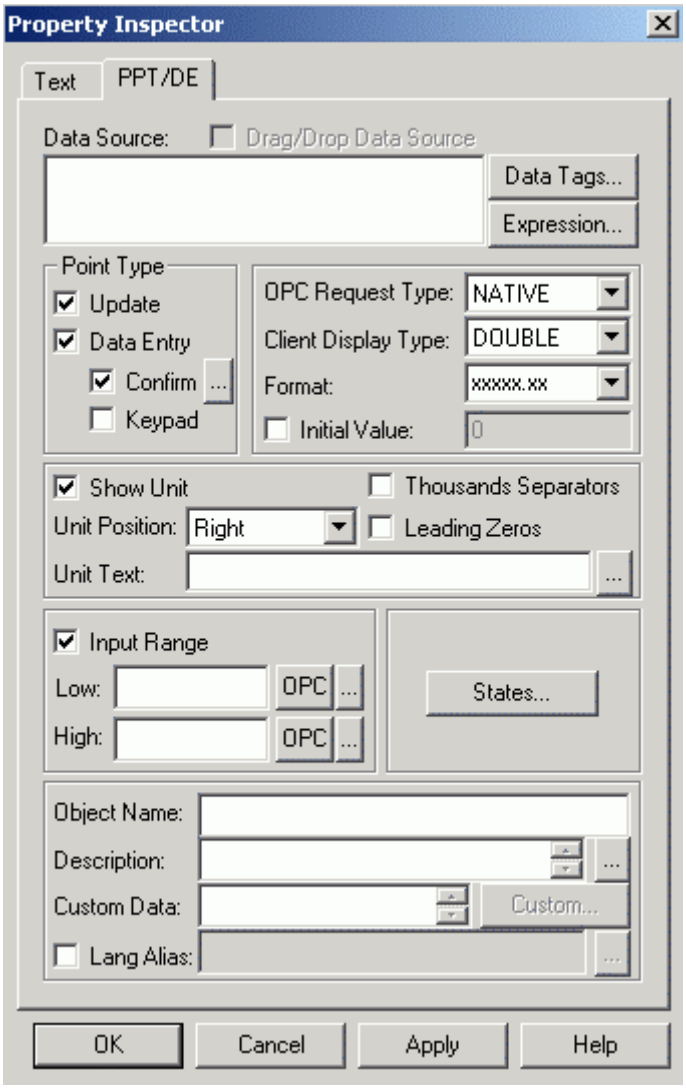
A **data entry** behaves the same way as a process point does, but a data entry also allows you to manually enter new values to be downloaded to the system.

To create a process point/data entry (PPT/DE) object:

1. Select **Intrinsics > Process Point** from the **Dynamics** menu, or click the **Process Point** button on the **Dynamics** toolbar.
2. Click inside the work area where you want to place the process point. This opens the **Process Point/Data Entry** dialog box, as shown below.
3. Select the parameters listed in the table below.

4. Additional properties relating to the visual appearance of the **PPT/DE** can be made by selecting the **Text** tab. The text properties for Process Points are similar to those used to edit static text objects.

5. Click **OK.** A box with question marks representing the PPT/DE format selected appears in the work area.



Property Inspector Dialog Box: Process Point Tab

Process Point/Data Entry Parameters

Parameter	Description
Data Source	Every Dynamic dialog has a data source. Data connections can be typed directly into the **Data Source** edit control field in the configuration dialogs of the various dynamics. Clicking the **Data Tags** button opens the **Unified Data Browser,** from which you can select tags in the system. You can also click the **Expression** button to create an expression using the Expression Editor. These methods of making data connections for the primary value are also available for making range override data connections.
Data Tags	Opens the **Unified Data Browser,** which enables you to select from OPC tags, data-mining tags, aliases, and variables. For more information, please see the Unified Data Browser help documentation.
Expression	Opens the **Edit Expression** dialog box, which enables you to create expressions. For more information, please see the Expression Editor help documentation.
Update	Defines a point type that the system updates in real-time.
Data Entry	Changes the value of the connected data point in runtime mode.
Confirm	Defines a confirmation message, which is a message the GraphWorX32 prompts every time the user writes a value. Click the **…** button to open the **Confirmation Message Builder,** which helps you create confirmation messages. For more information, please see the Confirmation Message Builder section below.
Keypad	Displays a keyboard or number pad to allow a value to be entered when the process point is clicked during runtime mode. You can set the size of the keypads using the **Runtime** tab of the **Application Preferences** dialog box.
OPC Request Type	The **OPC Request Type** is used in communicating OPC data between GraphWorX32 (an OPC client) and an OPC server, and between the OPC server and the I/O device(s) to which the OPC server is connected.
Client Display Type	The **Client Display Type** specifies how OPC data from an OPC server are shown in the GraphWorX32 display.
Format	Specifies the number of decimal places for the connected data point in runtime.
Float	Floating decimal point. Offers many default formats, such as "xxxx.xx," in a drop-down list for analog (floating point) process points. You can also define a scientific notation format using the predefined formats, or you can create a user-defined format. An example of scientific notation formats is the "x.xxe" format.
Double	Double precision. Offers the same formats as Float.

Bool	Boolean. Defines two-state display formats. Offers default formats, such as 0/1, Off/On, Yes/No, False/True, Auto/Manual, Enable/Disable, and Stop/Run.
Byte	8 bits. Offers the same formats as Short and Long.
Word	32 bits. Offers the same formats as Short and Long. "Word" is an unsigned 16-bit value.
DWord	64 bits (double word). Offers the same formats as Short and Long. "Double word" is an unsigned long value (32 bits).
Char	Character. Offers the same formats as Short and Long. "Char" is a signed 8-bit value.
Short	Short integer. Offers 10 default formats for integers, such as xxxx.
Long	Long integer. Offers the same formats as Short.
String	Any text string. Defines String text display formats, 10, 20, 30, 40, 50, 60, 70, and 80, as the number of characters. For String process points, you must define the number of text characters before you place this object in the display. Eight predefined string lengths are provided in the menu. You can use the predefined format or create your own.
Hexadecimal (Client Display only)	Hexadecimal constant. e.g. 0x20A = 2 * (16^2) + 0 * (16^1) + 10 * (16^0) = 2*256 + 0*16 + 10 * 1 = 512 + 0 + 10 = 522
Octal (Client Display only)	Octal constant. e.g. 0t36= 3 * (7^1) + 6 *(7^0) = 3* 7 + 6* 1= 21 + 6 = 27
Binary (Client Display only)	Binary constant. e.g. 0b110 = 1 * (2^2) + 1 * (2^1) + 0 * (2^ 0) = 1 * 4 + 1 * 2 + 0 * 1 = 4+2+0 = 6
Native (OPC Request only)	Unspecified data type.
Initial Value	Defines the initial (startup) value to which the data-entry object is initialized when the display is launched in runtime mode.
Show Unit	Check this check box to display a unit of measurement following (or preceding) the data value during runtime. The **Unit Position** can be left or right. The text can me anything (e.g.: “meters”). The **Unit Text** can also be a global alias or a language alias.
Thousands Separators	Check this check box to show thousands separators in runtime. The thousand-separator character (e.g. commas) depends on the regional setting. For example, in runtime the number “13200” (13 thousand two hundred) will appear as “13,200” in the United States and as “13.200” in Europe.
Leading Zeros	Check this check box to show leading zeros to the left of the data value in runtime; there will be as many leading zeros as required to reach the specified format length (number of decimal places). For example, if the format is

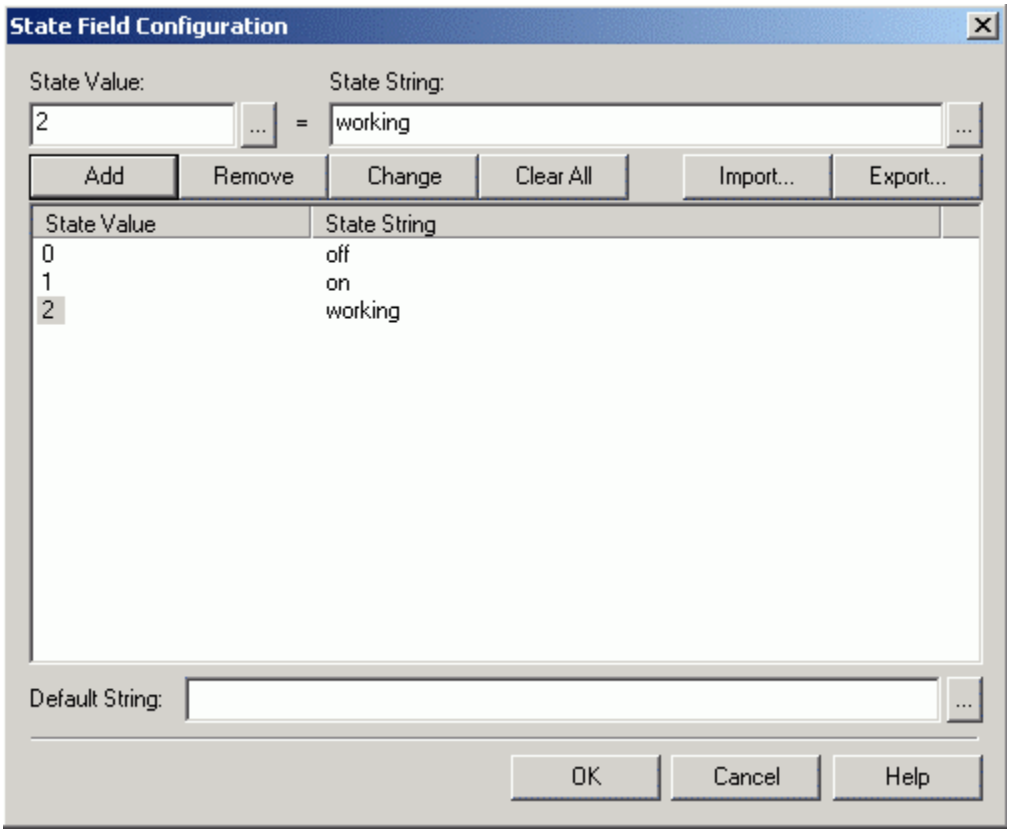
	"xxxx" and the value is "23," then the process point will show "0023."
Input Range	Specifies a low and high range in the boxes provided. This range applies to data entries only. In runtime mode, GraphWorX32 will not allow you to enter a value outside this range.
States	Opens the **State Field Configuration** dialog box, which allows you to enter the meaning of a binary value. See the **State Fields** section for more information.
Object Name	Identifies the object for OLE Automation. A name specified to this object can be referenced to identify this object. For example, this is the object name that can be used in VBA.
Description	Describes the object and usually coincides with what is displayed under ToolTips. **Note:** You can also select either **Global Aliases** or **Language Aliases** from the alias browsers by clicking the ... button to the right of the Description field.
Custom Data	Customized configured data. For example, this could be a custom configured data string. This will be enabled through the custom configuration utility tied to the **Custom** button. Refer to the **Custom Configuration** section for details.
Language Alias	Displays a language-aliased string (if applicable). Checking the **Lang Alias** check box and clicking the **...** button next to the **Lang Alias** box opens the **Language Alias Browser,** where you can select from all language aliases in the database.

State Fields

Clicking the **States** button in the **PPT/DE** tab of the Property Inspector opens the **State Field Configuration** dialog box, shown below, which allows you to enter the meaning of a binary value. **State fields** refer to the state of a variable. They are usually a binary value, which is attached to a string, which in turn has some meaning.

Note: The **Find / Replace** feature on the **Edit** menu is now supported for state fields. In the Find dialog box, select **Text Label** from the **Type** drop-down list, and then enter your search parameters.

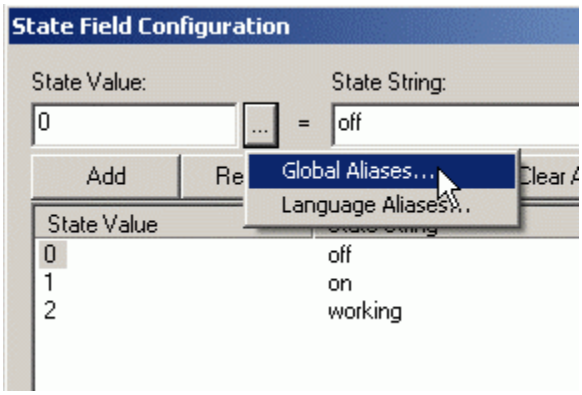
Note: The state field in GraphWorX32 is only able to display a single line of text.



State Field Configuration Dialog Box

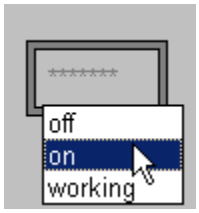
The configuration of state fields is very similar to that of aliases. You can enter a state value or simply select a state that already exists, enter or change the state string, and then click the proper action button (**Add, Remove, Clear** or **Clear All**). The **Import** and **Export** buttons open the **Open** and **Save As** dialogs, respectively. The **Default String** refers to the string that will appear when the state does not have a preconfigured string attached to it.

To select an alias, click the **...** button to the right of the **State Value, State String,** and **Default String** text boxes and select either **Global Aliases** or **Language Aliases** from the pop-up menu, as shown in the figure below. This opens the respective alias browsers, allowing you to choose from a list of available aliases.



Selecting an Alias To Use for State Fields

If the process point is set to **Data Entry,** a drop-down menu of all possible states will be available during runtime, as shown in the example below.



Data Entry Process Point In Runtime Mode

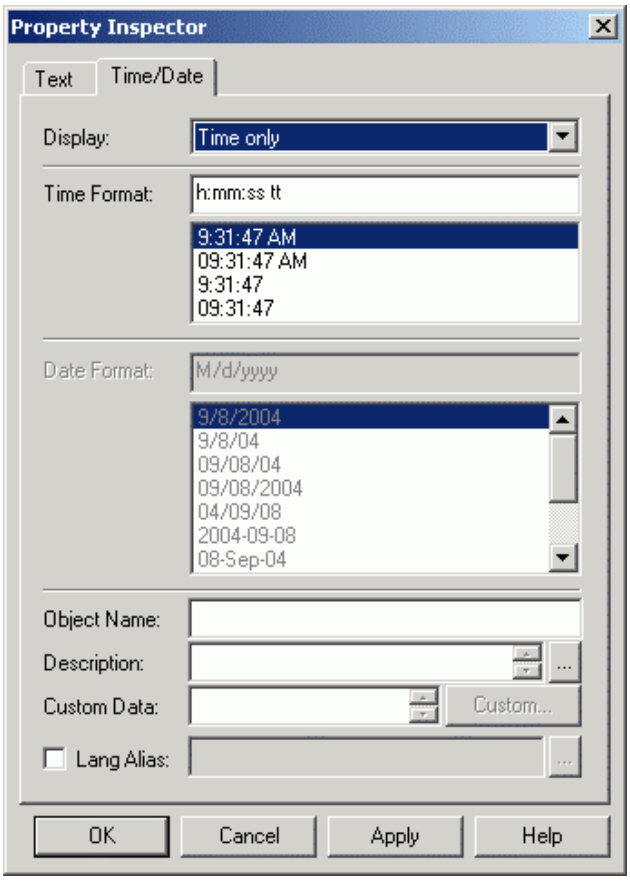
Time/Date



Different options are provided for the **Time** and **Date** functionality under the **Dynamics** menu of GraphWorX32.

To define **Time/Date:**

1. Select **Intrinsics > Time/Date** from the **Dynamics** menu, or click the **Time/Date** button on the **Dynamics** toolbar.
2. Click the work area where you want the Time/Date to appear.
3. This opens the **Time/Date** tab on the Property Inspector, as shown below.
4. Select a time and date format. Scroll through the various predefined formats in the format list box.
5. You can configure additional properties relating to the visual appearance of the Time/Date object using the **Text** tab, which is similar to that used to edit static text objects.
6. Click **OK.** The Time/Date object appears in your display.



Property Inspector Dialog Box: Time/Date Tab

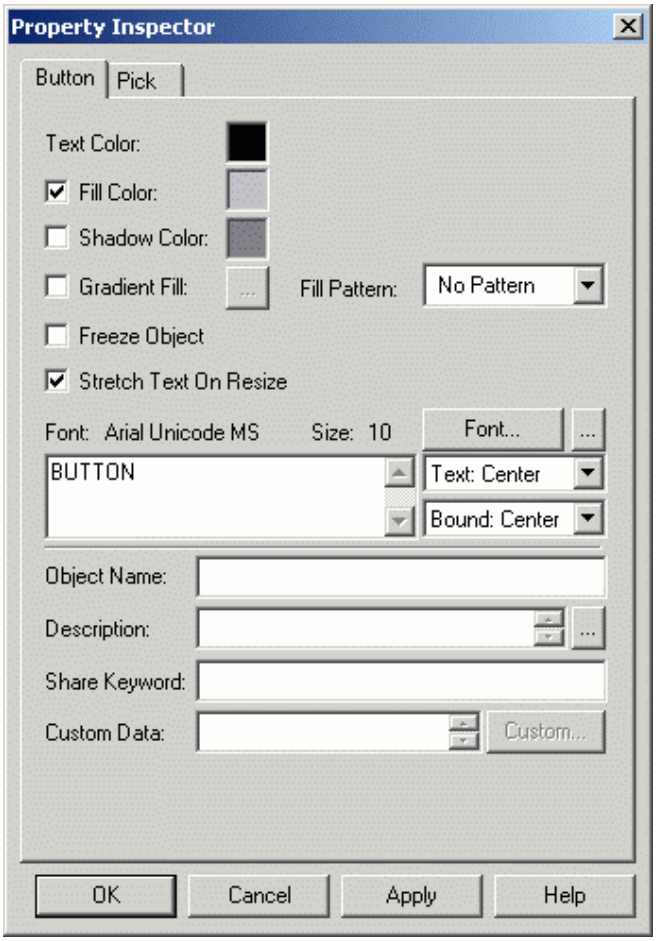
Push Button



The **Pushbutton** function creates a push button in a GraphWorX32 display. The push button object behaves almost exactly like the pick dynamic described earlier in this section. In addition to pick dynamic functionality, push buttons animate the up/down state in runtime mode.

To create a push button:

1. Select **Intrinsics > Pushbutton** from the **Dynamics** menu, or click the **Button** button on the **Dynamics** toolbar.
2. Click the work area where you want the pushbutton to appear.
3. This opens the property Property Inspector for pushbuttons, as shown below. The Property Inspector includes tabs for **Pick** (the same tab as the pick dynamic) and **Button** (the same tab as the static text object).



Property Inspector Dialog Box: Button Tab

4. Configure the properties for the push button, and then click **OK** to save these changes. The push button appears in the display, as shown below.

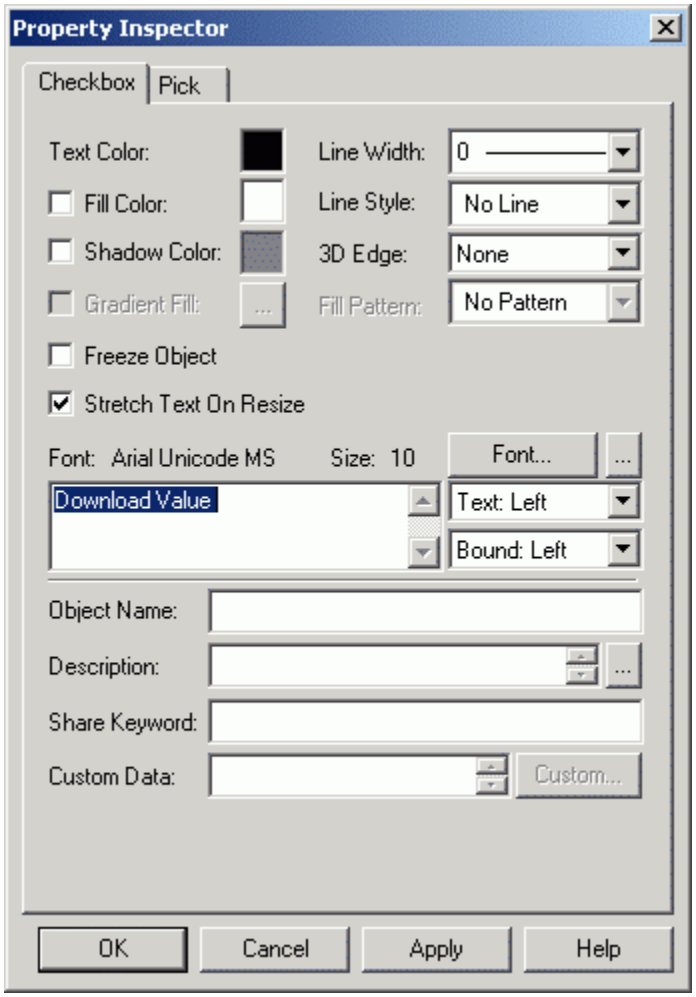
Check Box



The **Check Box** function allows you to create a check box in the same way you can create a push button. Check boxes differ slightly from normal push buttons in both appearance and behavior. GraphWorX32 check boxes look and act like standard Windows check box controls. When selected in runtime mode, the check box remains selected until it is clicked again.

To create a check box:

1. Select **Intrinsics > Check Box** from the **Dynamics** menu, or click the **Check Box** button on the **Dynamics** toolbar.
2. Click the work area where you want the check box to appear.
3. This opens the Property Inspector for check boxes. The Property Inspector includes tabs for **Pick** (the same tab as the pick dynamic) and **Check Box** (the same tab as the static text object).



Property Inspector Dialog Box: Check Box Tab

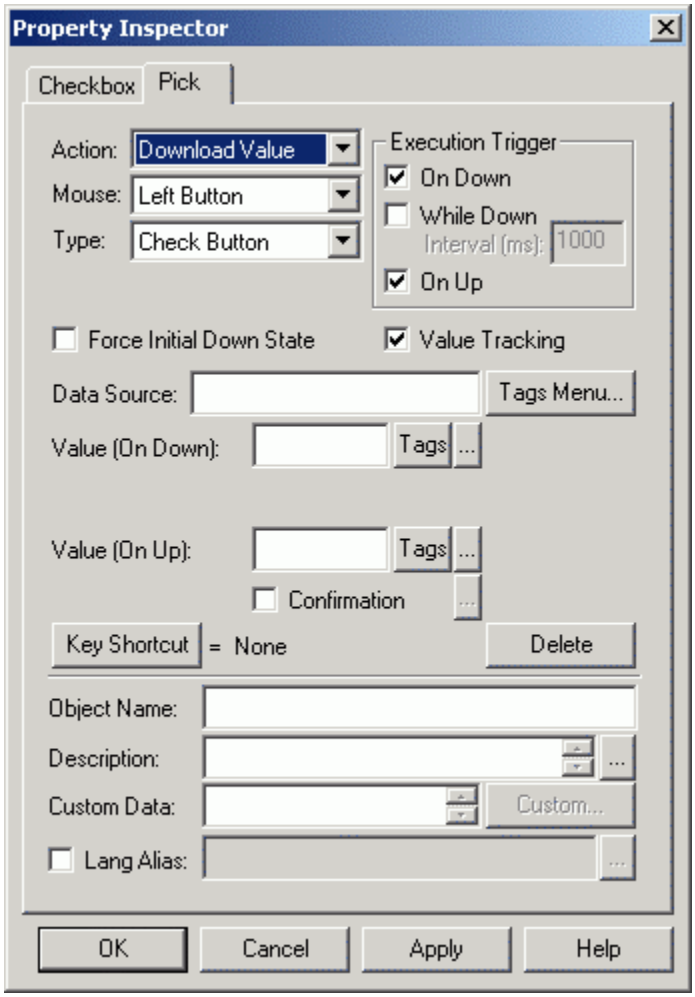
4. Configure the properties for the check box, and then click **OK** to save these changes. The check box appears in the display.

Tracking for Check Boxes

If the **Value Tracking** option is selected on the **Pick** tab of the Property Inspector, as shown in the figure below, then the check box that you are creating in your display will be checked or unchecked in runtime mode based on the connected data source tag value.

For example, create several check boxes in your display and connect them to the same tag (it can be a local variable). If the **Value Tracking** option is selected, a change made to one check box will be reflected by the other check boxes as well. Without the tracking option, the other check boxes do not change automatically.

Note: A similar example can also used for the radio button.



Value Tracking for Check Boxes

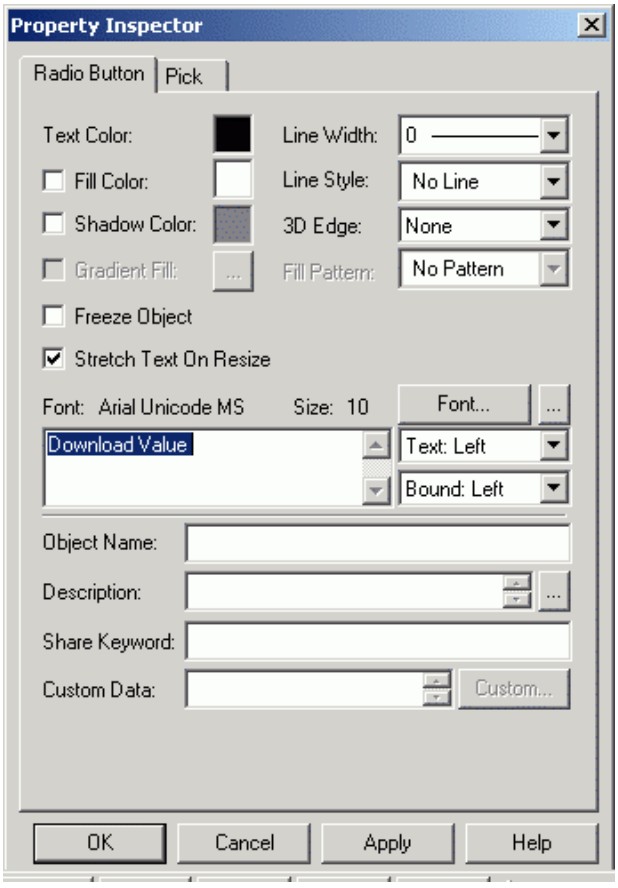
Radio Button



The **Radio Button** function allows you to create a radio button in the same way you can create a push button or a check box. Radio buttons differ slightly from normal push buttons in both appearance and behavior. GraphWorX32 radio buttons look and act like standard Windows radio buttons controls. When pressed in runtime mode, the radio button becomes selected and automatically deselects all other radio buttons that belong to the same group.

To create a radio button:

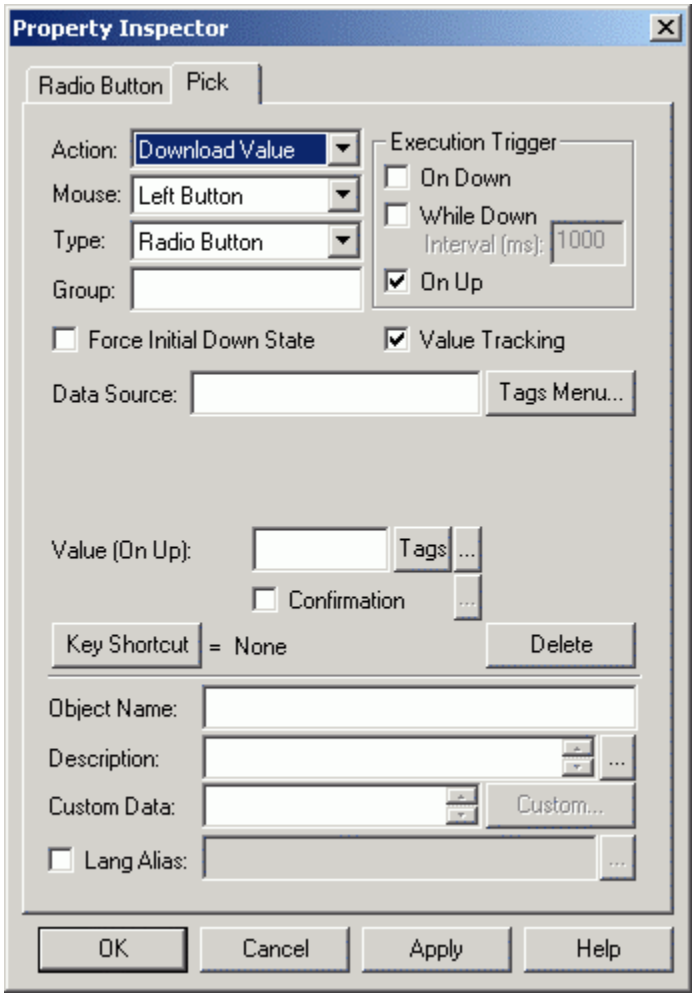
1. Select **Intrinsics > Radio Button** from the **Dynamics** menu, or click the **Radio Button** button on the **Dynamics** toolbar.
2. Click the work area where you want the radio button to appear.
3. This opens the Property Inspector for radio buttons. The Property Inspector includes tabs for **Pick** (the same tab as the pick dynamic) and **Radio Button** (the same tab as the static text object).
4. Configure the properties for the radio button, and then click **OK** to save these changes. The radio button appears in the display.



Property Inspector Dialog Box: Radio Button Tab

Tracking for Radio Buttons

If the **Value Tracking** option is selected on the **Pick** tab of the Property Inspector, as shown in the figure below, then the radio button that you are creating in your display will be selected or deselected in runtime mode based on the connected data source tag value.



Value Tracking for Radio Buttons

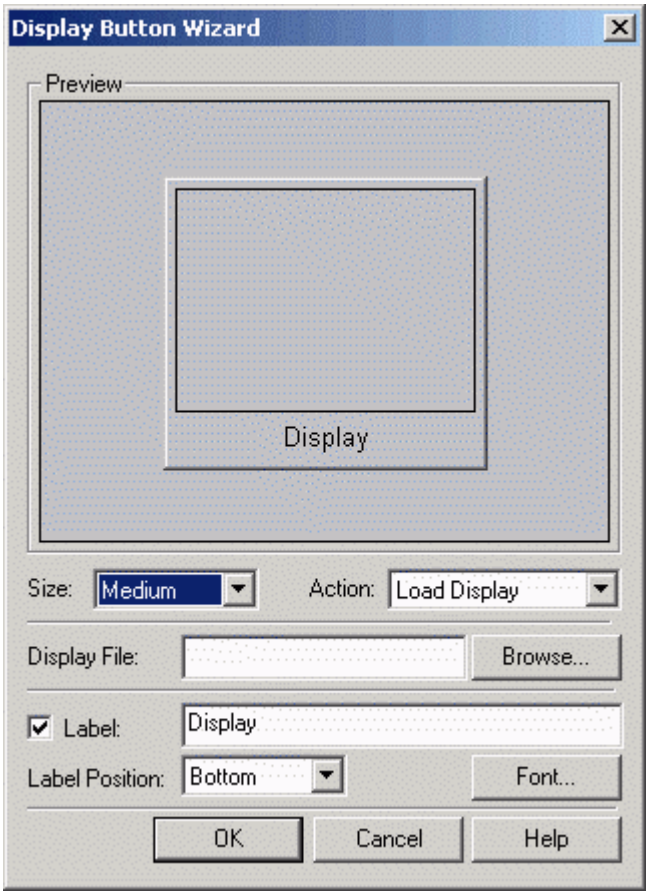
Display Button Wizard

GraphWorX32 contains a feature called a **display button.** A display button, once configured through a wizard, can do one of four things:

- Load a new display.
- Open a display in an embedded GraphWorX32 window.
- Display a popup window.
- Serve as a drag-and-drop object that can be placed in a GraphWorX32 ActiveX container to load a display.

This feature reduces clutter in a display because embedded windows and popup windows are shown only when needed by the push of a button.

To configure a display button, select **Intrinsics > Display Button Wizard** from the **Dynamics** menu, or click the **Display Button** button on the **Dynamics** toolbar. This opens the **Display Button Wizard,** shown in the figure below.

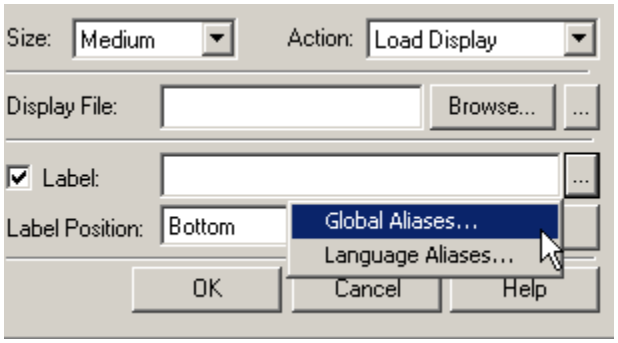


Display Button Wizard

The **Display Button Wizard** is a very simple, single-dialog wizard. The **Preview** field shows what the surface of the button will look like once it is configured. The options available for customizing both the appearance and functionality of the button are outlined in the following sections.

Note: Once the **Display Button Wizard** settings are changed, they will default to these settings for the remainder of the current editing session.

To select an alias, click the **...** button to the right of the **Display File** and **Label** text boxes and select either **Global Aliases** or **Language Aliases** from the pop-up menu, as shown in the figure below. This opens the respective alias browsers, allowing you to choose from a list of available aliases.



Selecting an Alias To Use for Display Buttons

Display Button Wizard - Size

The **Size** field in the **Display Button Wizard** allows you to select the size of the button from the drop-down list. The button in the **Preview** window resizes to match the item selected to show you exactly what size the button will be.

Display Button Wizard - Display File

In the **Display File** field in the **Display Button Wizard,** enter the name of the file that the display button will load. You can use the **Browse** button to search for and select a file. Once a file is selected, the button will show a shrunken image of the display it is configured to load.

To select an alias, click the **...** button to the right of the **Display File** text box and select either **Global Aliases** or **Language Aliases** from the pop-up menu, as shown in the figure below. This opens the respective alias browsers, allowing you to choose from a list of available aliases.

Display Button Wizard - Action

This combo box determines how the button will behave in runtime mode. Each of the four available actions is listed below and uses the display entered in the **Display File** field:

- **Load display:** Closes the current display and opens the new display.
- **Embedded window:** Opens the new display in an embedded GraphWorX32 window, leaving the original display open.
- **Popup window:** Opens a popup window that, unlike the embedded window, is not restricted to the inside of the GraphWorX32 display and can be maximized or minimized by the user.
- **Drag/drop load:** When this option is chosen, the button no longer behaves like a standard button. Rather it behaves like a drag/drop object. If there are existing GraphWorX32 containers in the display, the button can then be dragged and dropped into the container, causing the specified display to load.

Display Button Wizard - Label

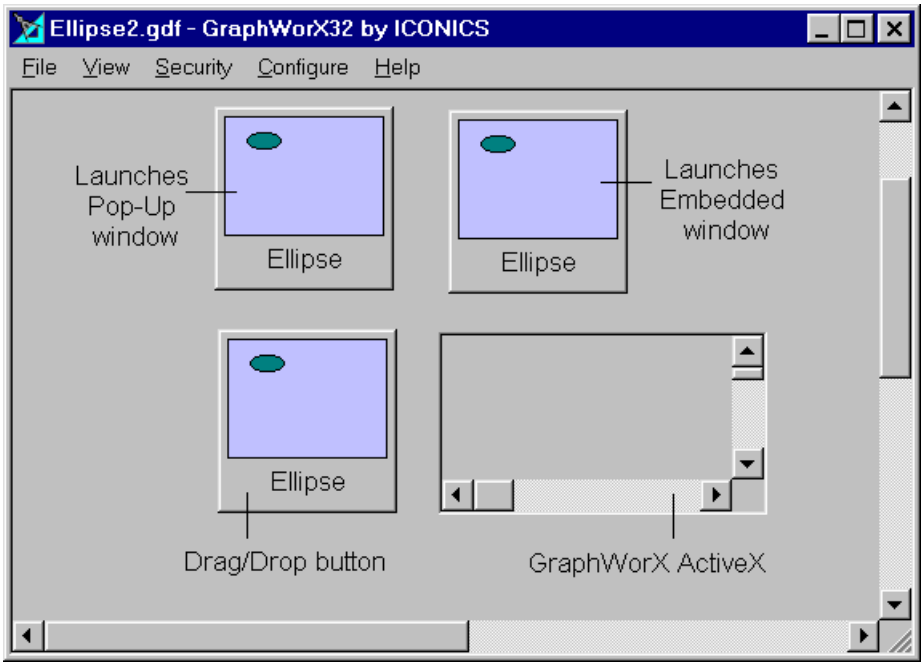
In the **Label** field of the **Display Button Wizard,** you can specify which label is to appear on the display button. If you do not wish to have a label on the button, uncheck the check box next to **Label.**

To select an alias, click the **...** button to the right of the **Label** text box and select either **Global Aliases** or **Language Aliases** from the pop-up menu, as

shown in the figure below. This opens the respective alias browsers, allowing you to choose from a list of available aliases.

Clicking the **Font** button to choose the font type. The appropriate attributes from the standard font window can change the label font. You can put the label near the top or near the bottom of the button by selecting either **Top** or **Bottom** in the combo box next to **Label Position.**

Once the wizard is complete, the display button is ready to be used in runtime mode. Upon execution, the button will perform according to the way it was configured in the wizard. The figure below shows three different display buttons. Each one launches a simple display called "Ellipse.gdf," which is simply an ellipse configured to rotate through the use of a simulated tag. One display button is configured to launch a popup window. The second launches an embedded window. The third is a Drag/Drop button located next to a GraphWorX32 ActiveX container.



Demonstration of Different Display Buttons in Runtime

Three different types of runtime windows, each with its own independently animated ellipse, all come into existence in runtime through the use of display buttons. At the top left of the figure is a demonstration of the effect of a display button used to launch a popup window. At the top right, an embedded window display button is shown. As you can see, the popup window can be minimized and maximized, while the embedded window cannot. At the bottom of the figure is a Drag/Drop display button next to a GraphWorX32 container that was, in configuration mode, an empty container. Dragging that button and dropping it into the container caused the ellipse display to load in the container. Had there already been a display in the container, it would have been replaced by the new display. In a more practical setting, multiple displays can be called by one master display through the use of display buttons. Entire displays get encapsulated in small, user-defined buttons ready to be activated at the user's request.

Custom Configuration

Every tab in the GraphWorX32 Property Inspector has a **Custom** button located next to the **Custom Data** edit field. This button can be tied to a custom configuration utility. The custom configuration utility has to be programmed by the user.

Clicking this button creates an Automation object that has a certain program ID (ProgID). By default the ProgID is **IcoCustom.Configure**. This ProgID can be overridden by setting an alternative ProgID in the Windows registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\Gwx32\OEM Information\Custom Configure ProgID = "***YourProgIDHere***"

You will probably want this Automation object to be exposed from a DLL. GraphWorX32 then uses the dispatch interface of this object to call the method:

boolean **CustomConfigure(**long hWndParent, IDispatch* ObjectOne, BSTR ObjectOneType, IDispatch* ObjectTwo, BSTR ObjectTwoType**)**;

Explanation:

hWndParent - the window handle of GraphWorX32 property inspector. You will probably want to popup a modal dialog as your custom configuration UI. Use this window handle as the parent of your dialog.

ObjectOne - dispatch pointer of the GraphWorX32 object associated with the Gwx32 property page currently shown (i.e. if the "pick" page is currently active, ObjectOne is the "GwxPick" object for that page).

ObjectOneType - String name of the object type for ObjectOne (for example, "GwxPick"). You will probably want to check the object type to be sure that you support configuration for this object type.

ObjectTwo - dispatch pointer of the GraphWorX32 Display to which ObjectOne belongs.

ObjectTwoType - String name of the object type for ObjectTwo (i.e. "GwxDisplay").

Return value is TRUE if you made changes to the objects passed to this method. FALSE if no changes were made (i.e. Cancel). DO NOT Release() the dispatch pointers passed to this method!!!

After the method returns, GraphWorX32 releases the dispatch pointer for the "IcoCustom.Configure" automation object.

Below is some sample code for a possible implementation of the **CustomConfigure** method:

```
BOOL CAutoConnect::CustomConfigure(long hWndParent,
LPDISPATCH ObjectOne, LPCTSTR ObjectOneType, LPDISPATCH
ObjectTwo, LPCTSTR ObjectTwoType)
{
  CString obj1Type(ObjectOneType);
  if (obj1Type == _T("GwxPick"))
  {
```

```
    CString strCustomData;
        //Run you configuration dialog here...
        if (returnValOfConfigDialog)
    {
      COleDispatchDriver obj1(ObjectOne, FALSE);
      //set user custom data
      obj1.SetProperty(0x10001, VT_BSTR,
            (LPCTSTR)strRCustomData);
      //set pick action = Custom Command
      obj1.SetProperty(0x1, VT_I4, (long)8);
      return TRUE;
    }
  }
  else
  {
    AfxMessageBox(_T("Custom Configuration for this object type is not
        supported."));
  }
  return FALSE;
}
```

Data and Timer Threads

The **Data Thread** and **Timer Thread** can be customized by two registry settings. If there is a need to increase the thread priority of GraphWorX, it is possible to configure these registry keys in the registry:

HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\Gwx32\Runtime Settings\DataThread\Priority

HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\Gwx32\Runtime Settings\TimerThread\Priority

The values are strings, and the following strings are valid:

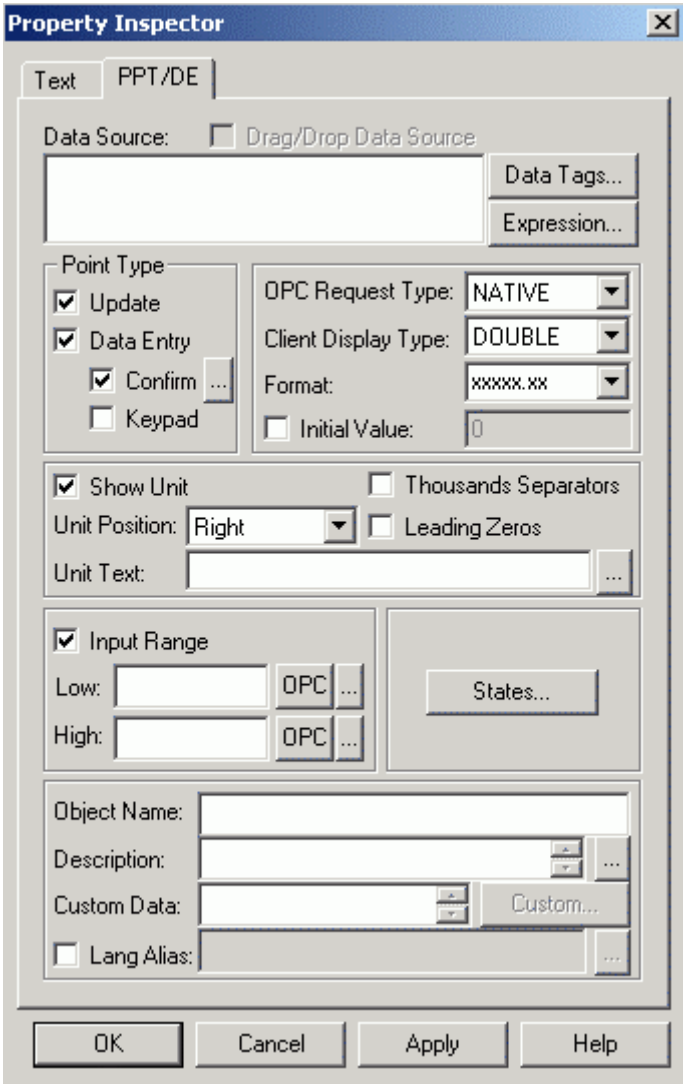
```
-15     THREAD_PRIORITY_IDLE
 -2     THREAD_PRIORITY_LOWEST
 -1     THREAD_PRIORITY_BELOW_NORMAL
  0     THREAD_PRIORITY_NORMAL  [default]
  1     THREAD_PRIORITY_ABOVE_NORMAL
  2     THREAD_PRIORITY_HIGHEST
 15     THREAD_PRIORITY_TIME_CRITICAL
```

Confirmation Message Builder

Several dynamic connection types items have a **Confirmation Message Builder** that helps you create confirmation messages that users will see when they write values to the GraphWorX32 display:

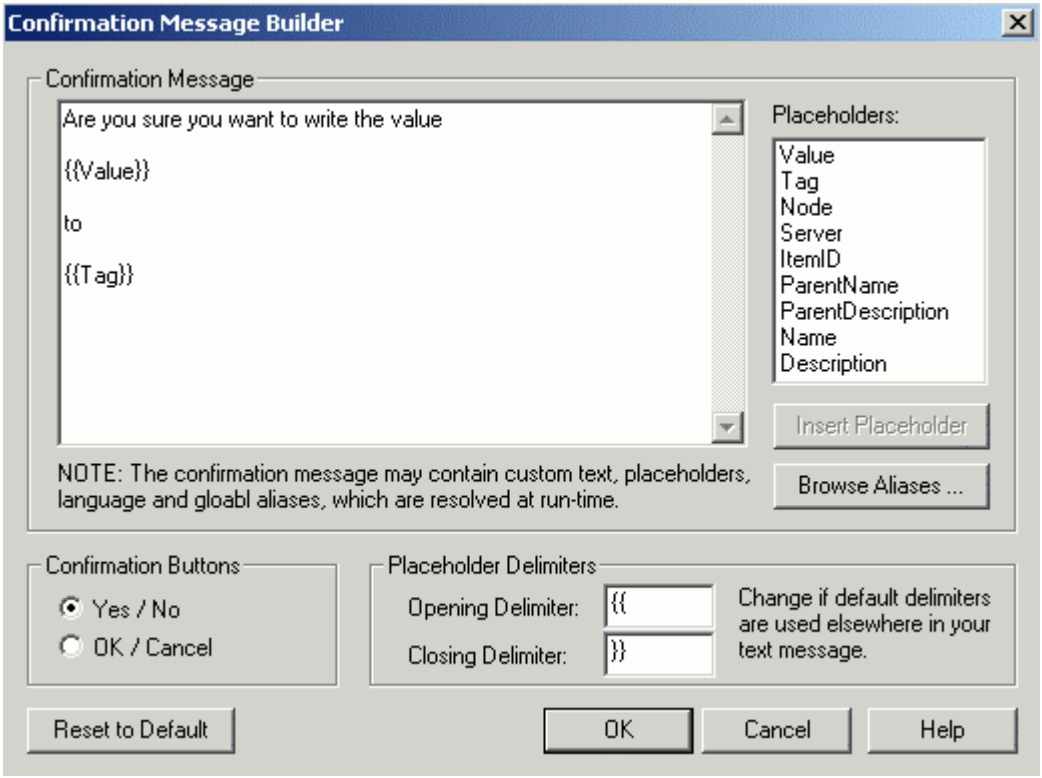
- Process Points and Data Entry
- Download Value pick action
- Toggle Value pick action
- Location/Slider
- Rotation Dial

For example, in the **Process Point/Data Entry (PPT/DE)** tab of the Property Inspector, shown in the figure below, when the **Data Entry** check box is checked, the **Confirm** check box is enabled. Checking the **Confirm** check box allows you to defines a confirmation message, which is a message the GraphWorX32 prompts every time the user writes a value to the process point.



Confirmation in Process Point

Click the **…** button to open the **Confirmation Message Builder,** shown in the figure below, which helps you create confirmation messages. The upper left side of the Confirmation Message Builder contains an edit box where you can compose your messages. The confirmation message consists of text, language aliases, global aliases, and placeholders, which represent a dynamic connection or a data item, such as a value, a tag, a name, etc.



Confirmation Message Builder

To select global and language aliases, click the **Browse Aliases** button and select from the respective alias browsers.

The upper right side of the dialog box conveniently provides several **Placeholders,** which you can use to write your confirmation messages. Simply select a placeholder and click the **Insert Placeholder** button (or double-click the placeholder), and it is inserted into the message box.

The example message shown in the figure above uses the following syntax:

"Are you sure you want to write the value {{Value}} to {{Tag}}"

Notice that **{{Value}}** and **{{Tag}}** are placedolders for the actual value and the tag name, respectively. By default, placeholder are delimited by **{{** and **}},** but you can change the opening and closing delimeters if these default delimiters are used elsewhere in your text message.

After you define your text message, select Confirmation Buttons that will prompt a user to confirm or deny the message. You can select either **Yes/No** buttons or **OK/Cancel** buttons, as shown in the figure above.

Data Mining and Data Visualization

GraphWorX32 enables data-mining of open database connections defined by the GENESIS32 Data Mining Configurator. Once you have configured your database connection, data sources, and data items in the Data Mining Configurator, the next step is to visualize the data in GraphWorX32. The Data Mining Demo provides an example GraphWorX32 display that shows how to configure your data mining connections so you can view data coming from the database.

Note: For more information, please see the Data Mining Configurator help documentation.

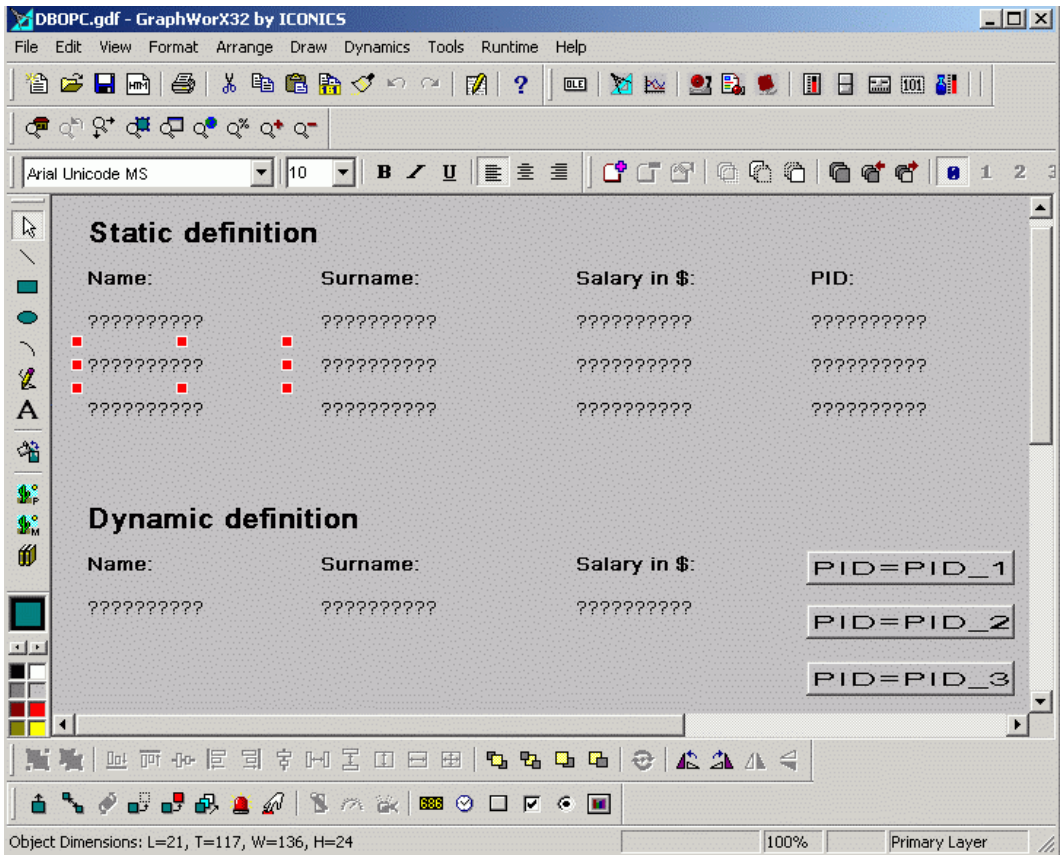
1. Go to the **C:\Program Files\ICONICS\GENESIS-32\Examples\GEN32DEMO\DatabaseMining** directory in the GENESIS32 installation and open the **DBOPC.gdf** file.
2. The display opens in GraphWorX32 configuration mode, as shown in the figure below. The display contains two different examples of data mining visualization: static and dynamic. A **static** display simply displays the data values from the connected database. A **dynamic** display allows you to manipulate data values from the connected database.

   The table in the **static definition** section contains some column names from the **Employees** database table, as shown in the figure below:

   - **Name:** Employee's first name
   - **Surname:** Employee's last name
   - **Salary:** Employee's salary
   - **PID:** Employee's identification number

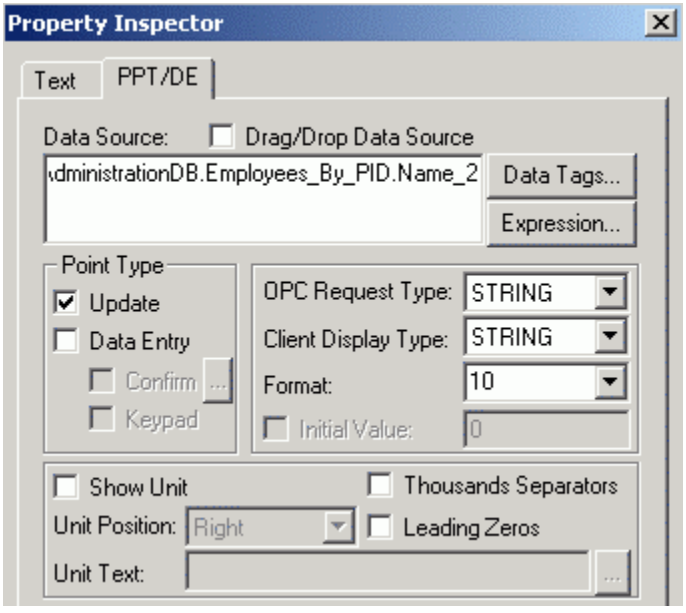
   Each column contains process points that correspond to data items (e.g. **Name_1, Name_2, Name_3, PID_1, PID_2,** etc.) in the **Employees_By_PID** data source from the demo data mining configuration. A process point must be configured for each data item. We will continue to look at the **Name_2** data item as an example:

   Under the **Name** column in the GraphWorX32 display, double-click the second process point under the **Name** column, as shown in the figure below.



Data Mining Demo in GraphWorX32: Configuration Mode

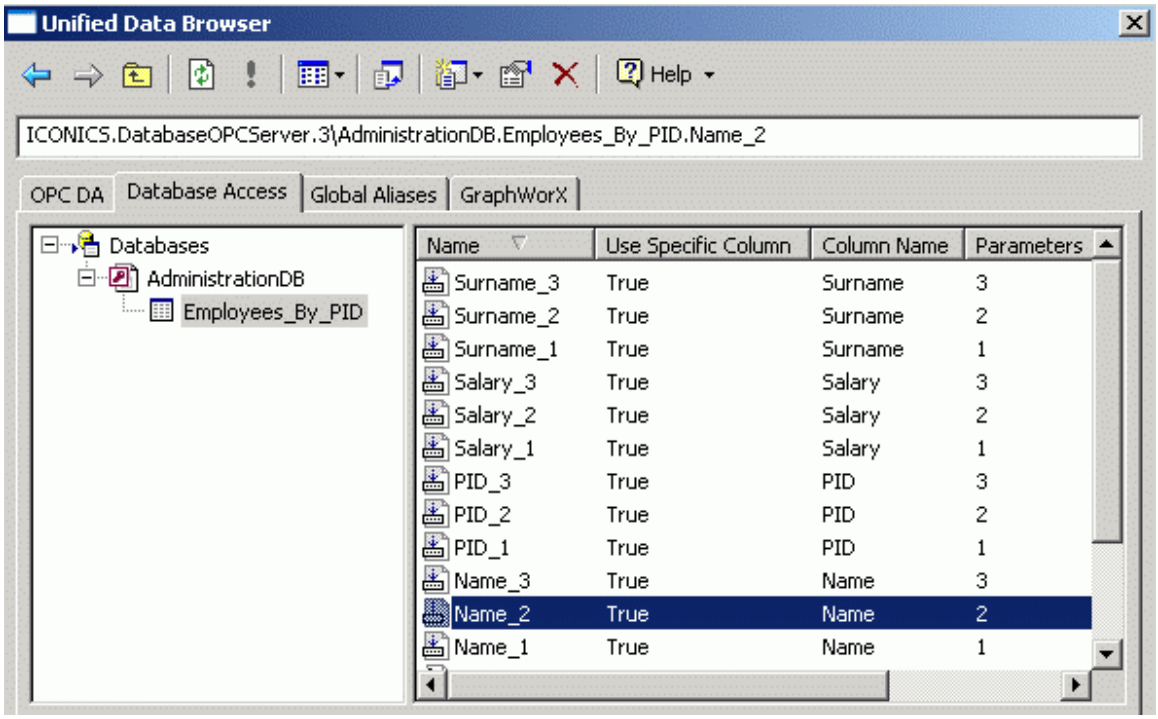
3. This opens the **Property Inspector** for the data item. Click on the **PPT/DE** tab, as shown in the figure below. As you can see, the **AdministrationDB.Employees_By_PID.Name_2** item ID has already been specified in the **Data Source** field.



Property Inspector for Name_2 Data Item

4. Click on the **Data Tags** button to open the Unified Data Browser. The Unified Data Browser opens to the **Database Access** tab, as shown in the figure below. You can see that the browser tree control is similar to the tree control in the Data Mining Configurator; you can choose from configured data items from the **Employees_By_PID** data source connection. The **Name_2** item ID has already been selected. Notice that each **ItemID** (e.g. **ICONICS.DatabaseOPCServer.3\AdministrationDB.Employees_By_PID.Name_2**) contains the following information:

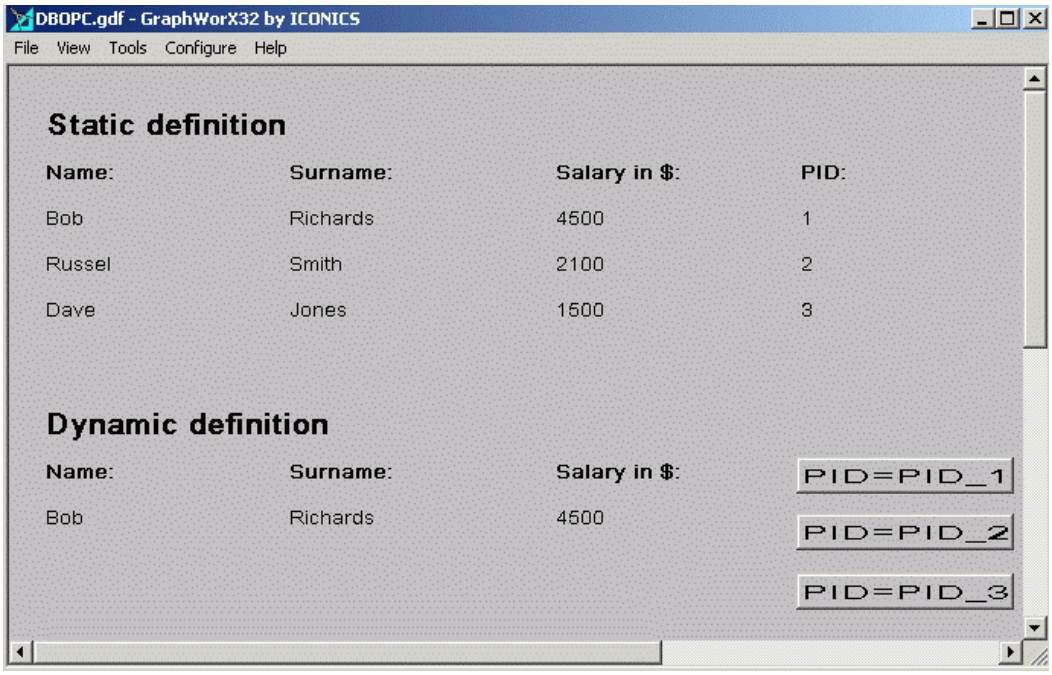
   - Server name (e.g. **ICONICS.DatabaseOPCServer.3**)
   - Database connection name (e.g. **AdministrationDB**)
   - Data source name (e.g. **Employees_By_PID**)
   - Data item name (e.g. **Name_2**)



Data Item Specified in Unified Browser

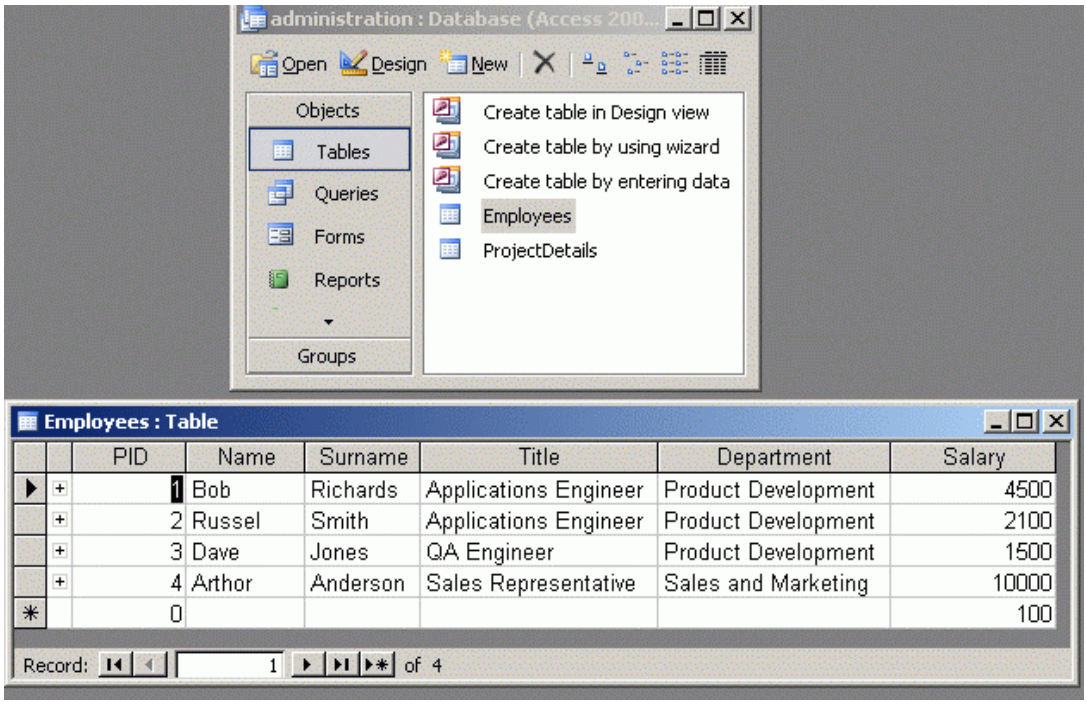
5. Close out of the Unified data Browser and the Property Inspector dialog box.
6. Enter the GraphWorX32 display into runtime mode. You can see the data coming from the **Employee** database table, as shown in the figure below. Notice that the **Parameter Value** for data item **Name_2** is resolved as "Russel" in the visualization display.

   Note: The **Static definition** section simply displays the values from the **Employee** database table for the configured data items, but the data cannot be manipulated. The **Dynamic definition** section also displays values from the **Employee** database table, but in this case only one row of data is displayed at a time, and you change the currently viewed row by clicking the three PID buttons, as shown in the figure below. The dynamic configuration uses local aliases and global aliases, which enable you to switch between rows during runtime mode.



Data Mining Demo in GraphWorX32: Runtime Mode

7. The items displayed in GraphWorX32 correspond to the information in the **Employees** database table in the Administration database. For example, the itemID **AdministrationDB.Employees_By_PID.Name_2** has the value **Russel,** because **Russel** is the second name listed in the database table, as shown in the figure below.



Employees Table in Administration Database

Chapter 9: Tools

Contents

Tools Menu
Function Keys
Set Working Directory
Security
Alias File Editor
Exporting to HTML
Download to Windows CE Support

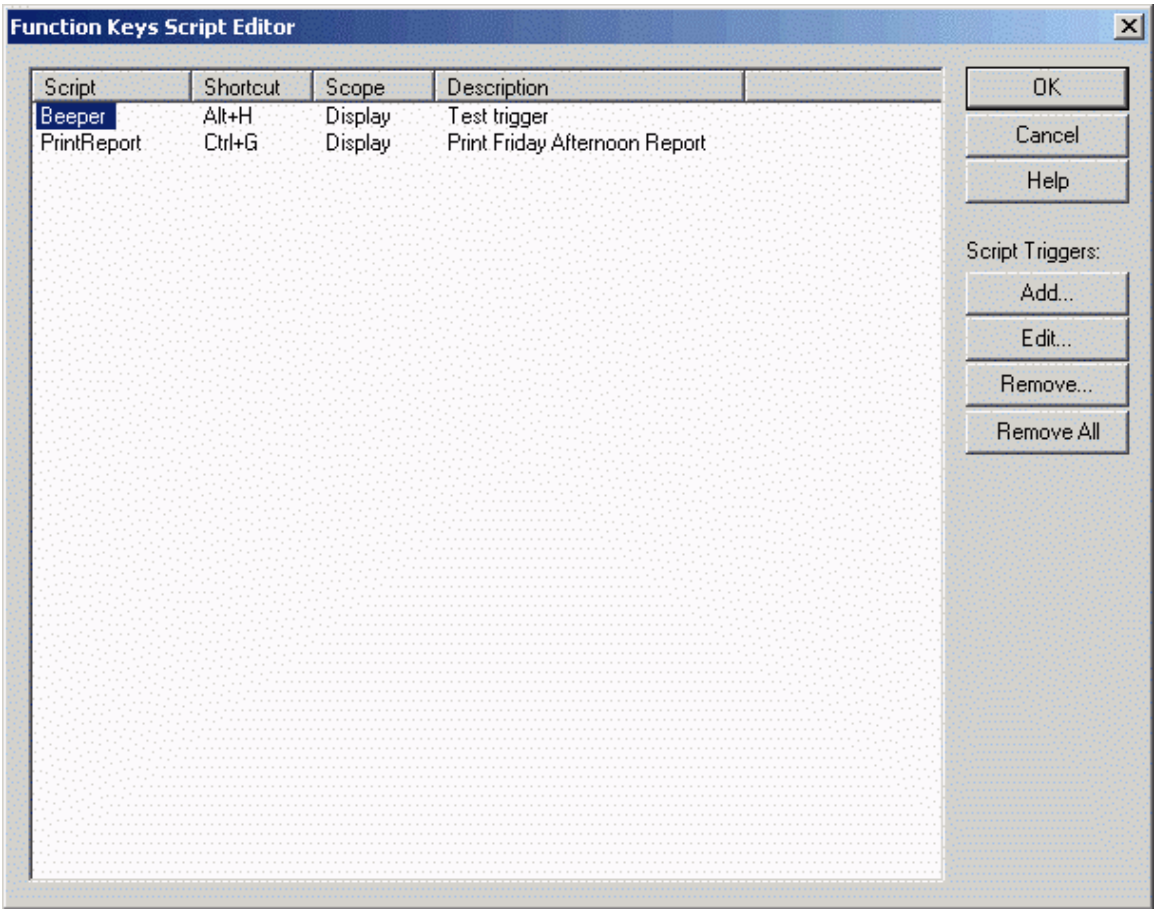
Tools Menu

The **Tools** menu allows you to access many tools that are used in the manipulation of GraphWorX32 data. The following is a list of options available from this list:

- Macros (Create, Edit, Step, Run, Clean Unused VBA Modules, and open the Visual Basic Editor). (For more information on the Visual Basic Editor, please refer to the Visual Basic for Applications documentation.)
- Function Keys
- Set Working Directory
- Security Configuration (For more information, please refer to the Security Configurator documentation.)
- Local Alias File Editor
- Global Aliasing Configuration (For more information, please refer to the Global Aliasing documentation.)
- Language Aliasing Configuration (For more information, please refer to the Language Aliasing documentation.)
- 16 bit to 32 bit Translator (For more information, please refer to the ICONICS GraphWorX32 Translation Utility documentation.)
- Publish to HTML
- Windows CE (Configure for Windows CE and Windows CE Preferences)

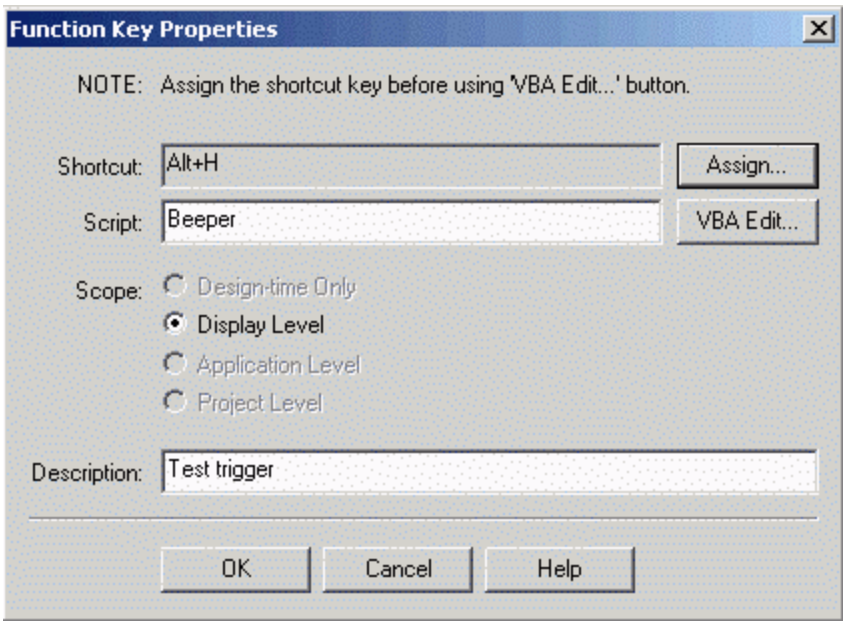
Function Keys

The **Function Keys** feature available on the **Tools** menu provides simple function keys management and a hierarchical functionality. Selecting **Function Keys** from the **Tools** menu opens the **Function Keys Script Editor,** shown below, which allows you to add, edit, and remove triggers. A **trigger** is a combination of a shortcut key and a VBA script name, which together allow you to open a VBA script when the shortcut key is pressed.



Function Keys Script Editor

Clicking the **Add** button or the **Edit** button opens the **Function Key Properties** dialog box, shown below, which enables you to specify the shortcut and the script.



Function Key Properties

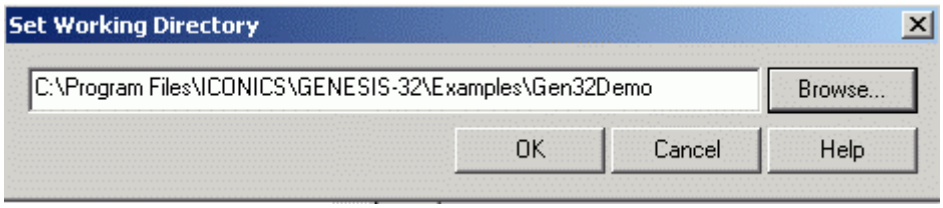
You can either type the script name or click the **VBA Edit** button, which creates a script or jumps to the script if the script already exists. You can now type the script body in the VBA editor. It is recommended that you debug the script and verify that it compiles without problems by clicking **Debug** and **Compile Project** in the VBA Editor.

The **Scope** field specifies the scope of the script. Currently only the **Display Level** script scope is available. Display level function keys are the first step of the function keys hierarchy. The other scope options are disabled.

The **Description** field is simply a text field that appears in the last column of the **Function Keys Script Editor.**

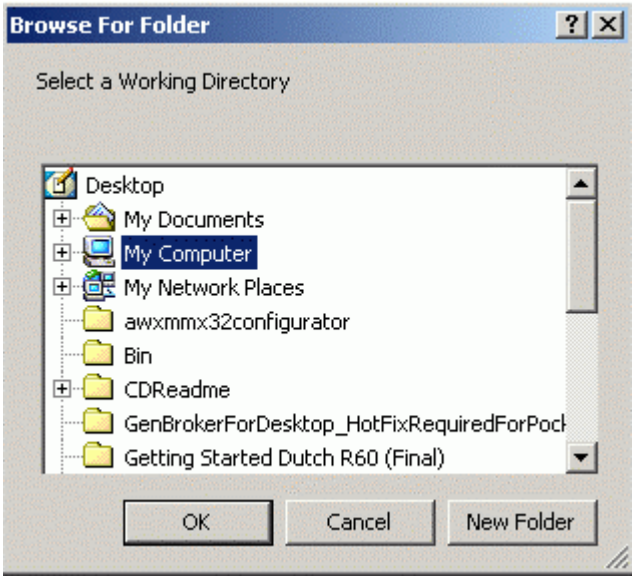
Set Working Directory

Selecting **Set Working Directory** from the **Tools** menu opens the **Set Working Directory** dialog box, shown below, which enables you to configure a custom directory in which all application configuration files will be stored and retrieved.



Set Working Directory Dialog Box

Click **Browse** to select the directory from the **Browse for Folder** dialog, as shown in the figure below.

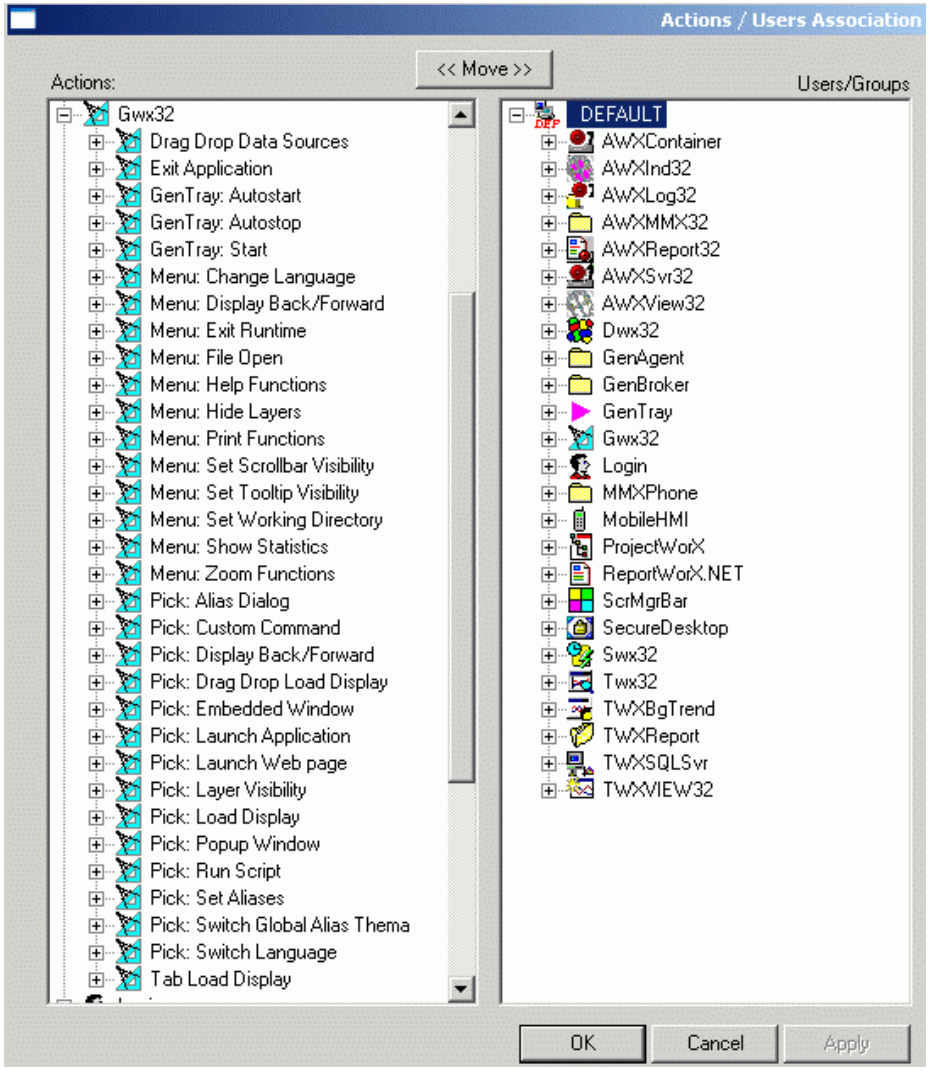


Selecting a Working Directory

Security

GraphWorX32 has an interface with the ICONICS Security Server and currently supports the security actions shown in the dialog box below.

For more information about security, please refer to the Security Configurator help documentation.



GraphWorX32 Security Actions

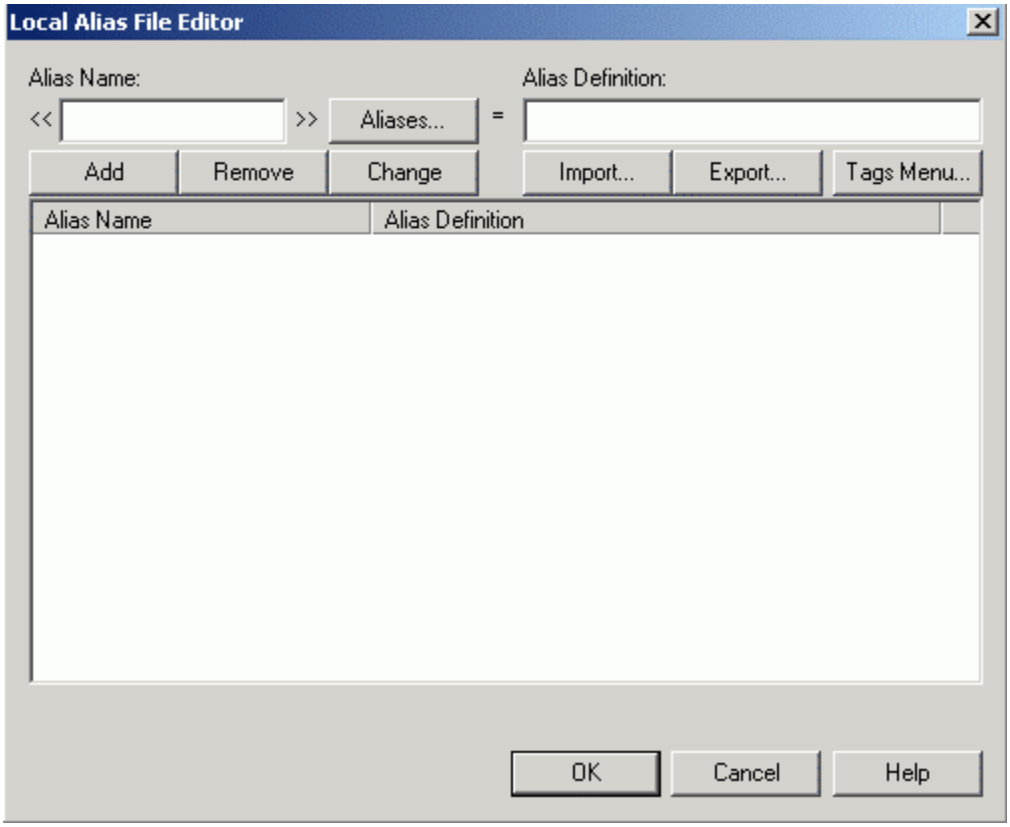
Disabling the Security Messages and Login application

The following registry options allow you to disable the security message box “Insufficient security…” and the login.exe application:

HKLM / GWX32 / Compatibility Properties / SuppressSecurityLogin
HKLM / GWX32 / Compatibility Properties / SuppressSecurityMessageBox

Local Alias File Editor

The **Local Alias File Editor** command on the **Tools** menu opens the **Local Alias File Editor** dialog box, shown below, which allows you **Add, Remove,** and **Change** aliases. Click on the **Aliases** button to show aliases for the display. You can also choose to **Import** or **Export** an alias. Under the **Alias Definition** field, you can define aliases using the **Tags Menu** button, which allows you to select tags, expressions, local variables, simulation variables, and global aliases.



Local Alias File Editor Dialog Box

Publishing to HTML

The GENESIS32 **Web Publishing Wizard** enables you to "export" your GraphWorX32 (.gdf), TrendWorX32 (.t32), and AlarmWorX32 (.a32) displays to HTML files and/or publish the HTML files to a Web server (LAN or Internet). In publishing displays to a Web server, WebHMI uses HTML to reference the files in an Internet-enabled format. Once a display is "exported" to an HTML file and then published to a Web server, client machines can browse it through an Internet browser, such as Microsoft Internet Explorer. Each display can be viewed as a Web page.

Basic Steps in Publishing GraphWorX32 Displays

Publishing a GraphWorX32 display to HTML involves the following basic steps:

1. Create a GraphWorX32 display (.gdf) file on a developer workstation with GraphWorX32 installed.
2. Use the Web Publishing Wizard to generate an HTML file and publish the file to a Web server (URL) address.
3. On the Web server PC (with GENESIS32 installed), start GenBroker Server from GenTray.
4. On a client PC, open a Web browser, such as Microsoft Internet Explorer, and browse to the URL address of the HTML file. The client PC downloads the HTML file from the Web server. The Web server delivers all the necessary components (e.g. ActiveX controls) to the client PC's Web browser, which then runs the GraphWorX32 display directly within the browser. This way you can view the GraphWorX32 display in real time as a Web page.

The source code of the published HTML file contains references to the required “plug-ins” that are needed to deliver the GraphWorX32 Viewer ActiveX control to the client. The GraphWorX32 Viewer ActiveX is needed to run the .gdf file over the network. The GraphWorX32 display itself is not really “converted” into HTML. Instead the display (.gdf) file is referenced in the HTML code by the ActiveX plug-in.

To expand on step 2 above, the following Microsoft Internet Explorer example shows the portion of the generated HTML source code that references the .gdf file. The **GWXview32.cab** file is referenced in the **codeBase** field to deliver the “plug-in” for the GraphWorX32 Viewer ActiveX. This in turn opens up the .gdf display. In the example below, the .gdf display file is called **Building1.gdf,** as shown below in the **DisplayName** parameter field.

```
<OBJECT classid=clsid:98A5DDE3-563B-11CF-A343-487C03C10000
codeBase="http://www.myserver.com/webhmi/cabs/GWXview32.cab"
id=GWXview321 style="HEIGHT: 420px; WIDTH: 620px" height=420
width=620>
<param name="_Version" value="65537">
<param name="_ExtentX" value="16404">
<param name="_ExtentY" value="11113">
<param name="_StockProps" value="160">
<param name="BorderStyle" value="1">
<param name="Appearance" value="1">
<param name="DisplayName"
value="http://www.myserver.com/webhmi/Building1.gdf">
<param name="UseAmbientBackColor" value="0">
<param name="AutoStartRuntime" value="-1">
<param name="OverrideScrollbarSettings" value="1">
```

```
<param name="VerticalScrollbar" value="0">
<param name="HorizontalScrollbar" value="0">
</OBJECT>
```

Exporting a .gdf file to an HTML file not only references the GWXview32.cab file, as shown in the example above, but also references any other "plug-in" .cab files required for other ActiveX components that may be embedded within the .gdf file (e.g., TrendWorX32 or AlarmWorX32 Viewer ActiveX controls).

However, before a .gdf file can communicate with live OPC data, the **IcoSetServer.cab** file “plug-in,” which is necessary for security and licensing, as well as a **GenBroker configuration (.gbx) file,** which contains network configuration settings for OPC communications, must also be delivered to the client PC. The IcoSetServer.cab file and the .gbx file are referenced in the HTML source code. This way the security and licensing information are available whenever a component (such as an Alarm Viewer ActiveX) is downloaded. The sample HTML source code below shows how these files are referenced in the code.

```
<object id="SetServer" classid="clsid:57802C16-9A15-11D4-B2A8-
0090272E599B"
codeBase=http://www.myserver.com/WebHMI/cabs/IcoSetServer.cab
height=28 width=17>
   <PARAM NAME="CfgName"
VALUE="http://www.myserver.com/WebHMI/Samples/Default.gbx">
</object>
```

The GENESIS32 Web Publishing Wizard in GraphWorX32 takes care of all the necessary HTML code references automatically. Microsoft Internet Explorer uses .cab files.

Note: If you are using multiple frames for your WebHMI pages (e.g. a main "navigation" frame for browsing between pages, and a "content" frame that contains the body of the pages), make sure that the IcoSetServer.cab file and the .gbx file are properly referenced in the HTML source code for the main frame. This way the security and licensing information are available whenever a component is downloaded.

Delivering the Necessary Web Components to the Client

WebHMI is designed to operate with **Zero Install** and **Thin Client** philosophy. This means the client PC has nothing but Windows and Internet Explorer loaded. Thus, all necessary Web components must be remotely delivered and seamlessly installed. The type and number of components required for delivery on the client PC are determined by the content of the GraphWorX32 displays. For instance, a display with an embedded Trend Viewer requires delivery of the corresponding TrendWorX32 Viewer ActiveX Web component, but not the AlarmWorX32 Viewer ActiveX. On the other hand, a display with an embedded Alarm Viewer requires the corresponding AlarmWorX32 Viewer ActiveX but not the TrendWorX32 Viewer ActiveX.

The delivery process can be very slow in terms of network speed. To avoid wasting time, the first requirement is to deliver only the required component. The delivery process is strictly connected to the HTML code. Inside the HTML code, there are references to .cab files, the basic ways to deliver components. As a result, the building of the HTML page is a critical step. The Web Publishing Wizard looks inside the display to understand what components are needed on the client side and then builds the corresponding HTML code.

Client Station Requirement

In order to view a published GraphWorX32 display, a client PC must have Microsoft Internet Explorer installed.

Developer Station Requirement

The developer station requires GraphWorX32.

Web Server Station Requirement

The Web server PC must have the following installed:

- WebHMI
- GENESIS32 (with GenBroker)
- For Windows NT, Internet Information Server (IIS) or a Personal Web Server
- For Windows 2000, Windows XP and Windows Server 2003, Internet Information Server (IIS)

  Note: Because publishing via Microsoft Posting Acceptor is no longer supported on Windows Server 2003, we were forced to find another way of transporting published files from a GENESIS32 development machine to a WebHMI server. It appeared that FTP would be the best choice. For more information, please see the "Publishing Channel" section below.

Multiple Display Support

Suppose that you have a main GraphWorX32 display (.gdf) file that is linked to other display files (e.g. each display contains pick actions, such as Load Display or Pop-up Window, that point to the other display files). When you publish the main display to an HTML file, you want all the links and references to the other dependent display files to be functional when the HTML file is downloaded to a client Web browser. The Web Publishing Wizard makes this possible by detecting all mutually linked display files, looping through all dynamic actions, and checking for pick actions in which a file name is specified as one of its parameters. The following pick actions are supported for multiple GraphWorX32 display Web publishing support:

- Load Display
- Embedded Window
- Popup Window
- Drag/Drop Load
- Set Aliases
- Alias Dialog (alias files specified through the **Set Aliases** dialog box in GraphWorX32).

The LoadTabs display property is also supported for publishing multiple GraphWorX32 displays.

Embedded ActiveX Control Support

In dealing with a main GraphWorX32 display (.gdf) file that is linked to other display files (as described in the "Multiple Display Support" section above), suppose you included an ActiveX Control (e.g. Alarm Viewer ActiveX or Trend Viewer ActiveX) into each of the dependent display files. Also suppose that you generated and published an HTML file only from the main display, but you want this single HTML file to trigger the simultaneous download and installation of all ActiveX Controls embedded within all interlinked GraphWorX32 display files. The Web Publishing Wizard's embedded ActiveX Control support makes this possible, allowing you to view the ActiveX Controls in all displays from a single client Web browser.

Using the Web Publishing Wizard

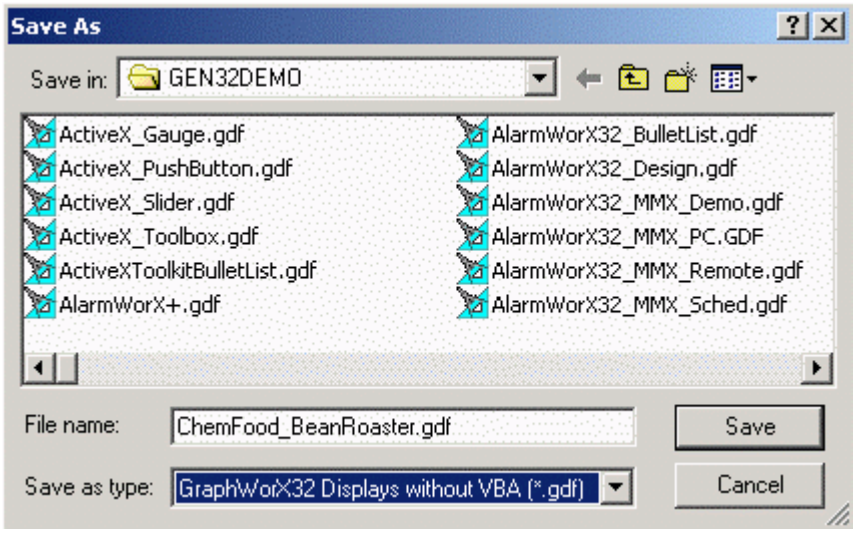
The Web Publishing Wizard performs two basic operations:

1. The Wizard creates an HTML file based on a user-specified GraphWorX32 display (.gdf) file, TrendWorX32 display (.t32) file, or AlarmWorX32 display (.a32 file).
2. The Wizard then either "exports" (saves) the HTML file to a user-specified directory on the local drive and/or "publishes" (uploads) the HTML file to a user-specified Web server URL address (i.e. over the Internet or an intranet).

Starting the Web Publishing Wizard

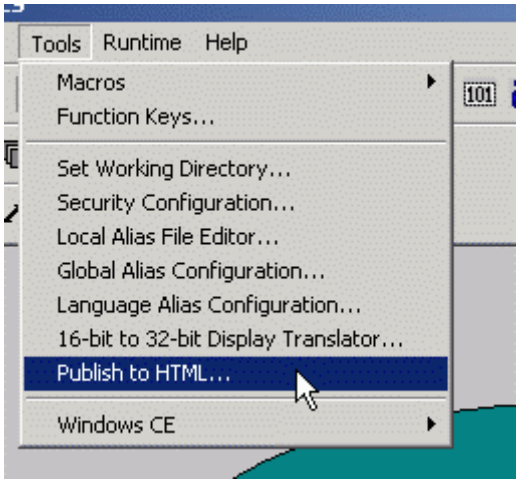
To launch the Web Publishing Wizard in GraphWorX32:

1. Load or create a GraphWorX32 display (.gdf) file.
2. After you have either created a new display file or opened an existing file, select **Save As** from the **File** menu. The **Save As** dialog box opens, allowing you to specify the name and location of the file you are about to save. Type a name for the new file in the **File Name** field. It is important that, when saving your file, you select "GraphWorX32 Displays without VBA (*.gdf)" in the **Save as Type** field, as shown in the figure below, because the Web Publishing Wizard does not support Microsoft Visual Basic for Applications (VBA). Click the **Save** button to save the current file.



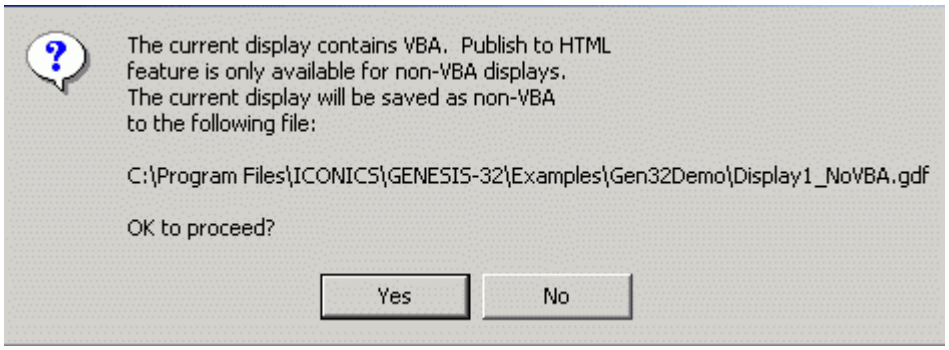
Saving the Display As a Non-VBA File

3. Select **Publish to HTML** from the **Tools** menu, as shown in the figure below.



Starting the Web Publishing Wizard in GraphWorX32

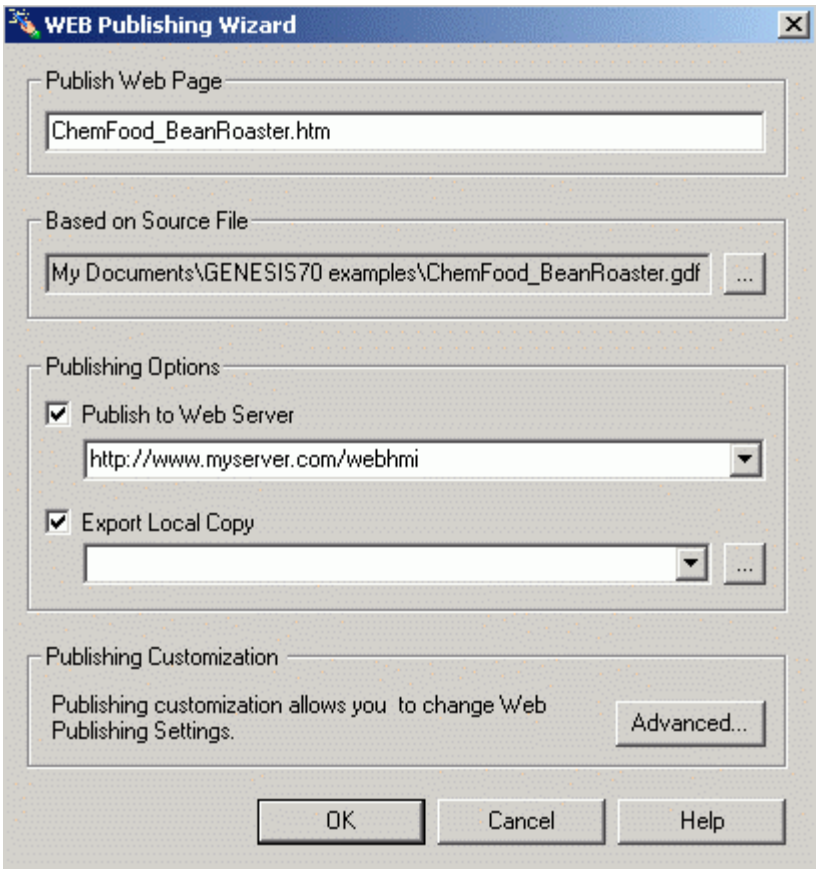
Note: If you try to run the Web Publishing Wizard with a GraphWorX32 display with VBA, you will get a warning message as shown in the figure below. If you click **Yes,** the Web Publishing Wizard automatically saves the display as a non-VBA file. If you click **No,** the publishing operation is cancelled.



Warning Message About Displays With VBA

4. This launches the **Web Publishing Wizard** configuration dialog box, as shown in the figure below. This dialog box serves as the interface through which you export/publish GraphWorX32 display files to the Web.

Note: Both the **Publish to Web Server** and **Export Local Copy** actions can be performed at the same time.

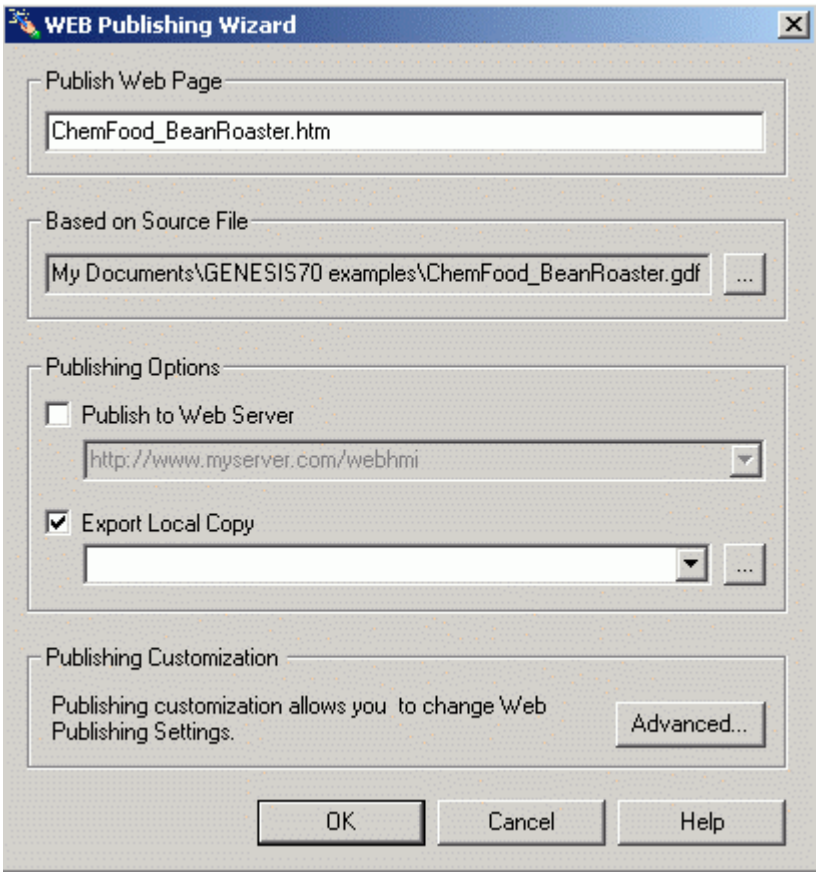


Web Publishing Wizard Dialog Box

Exporting a Display File Locally

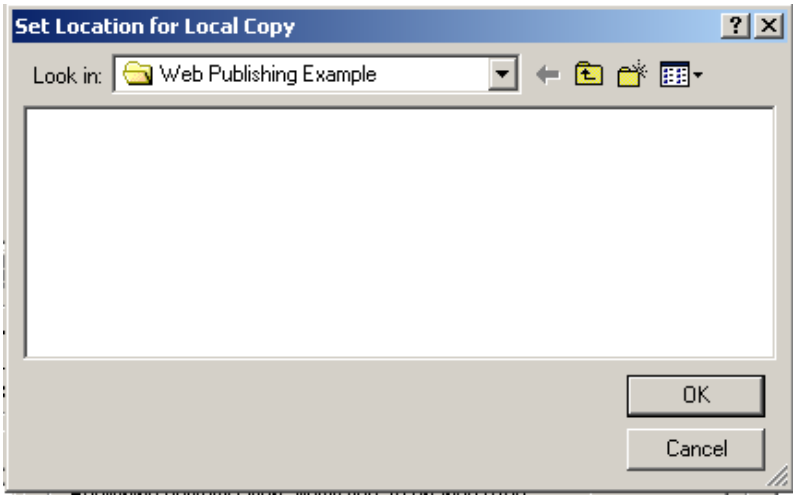
To export a GraphWorX32 display to a directory on the local drive:

1. In the **Based on Source File** field of the Web Publishing Wizard, you must specify the name of the GraphWorX32 display (.gdf) file to be exported, as shown in the figure below. To choose a different display, click the **…** button to the right to browse for a file. Select a file and then click **Open.** The directory path and the file name are shown in the text field.



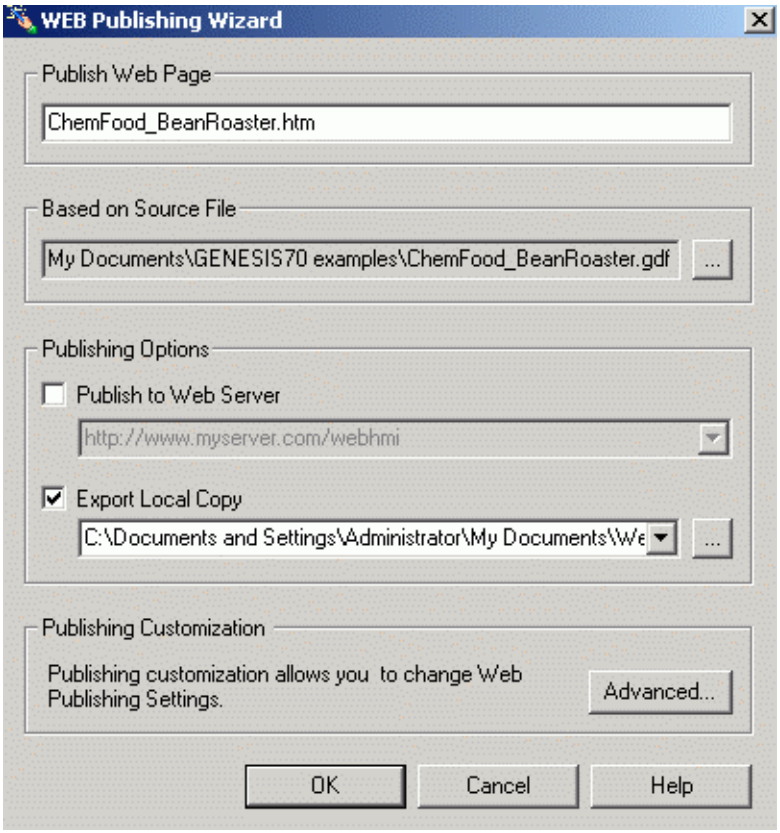
Specifying a GraphWorX32 Display File

2. In the **Publish Web Page** field of the Web Publishing Wizard, specify the name of the HTML file that will be created. The .gdf file name is filled in by default, but you can give the HTML file a different name.
3. In the **Publishing Options** field, check the **Export Local Copy** check box and specify the local directory path name to which you want to export the HTML file. You can select a recently used path from the drop-down list, or click the **…** button to browse for a destination directory, as shown in the figure below. Select the directory and click **OK.**



Setting the Local Directory

4. The local directory pathway you selected appears in the **Export Local Copy** field of the Web Publishing Wizard, as shown in the figure below.



Export Local Copy Pathway Shown in the Wizard

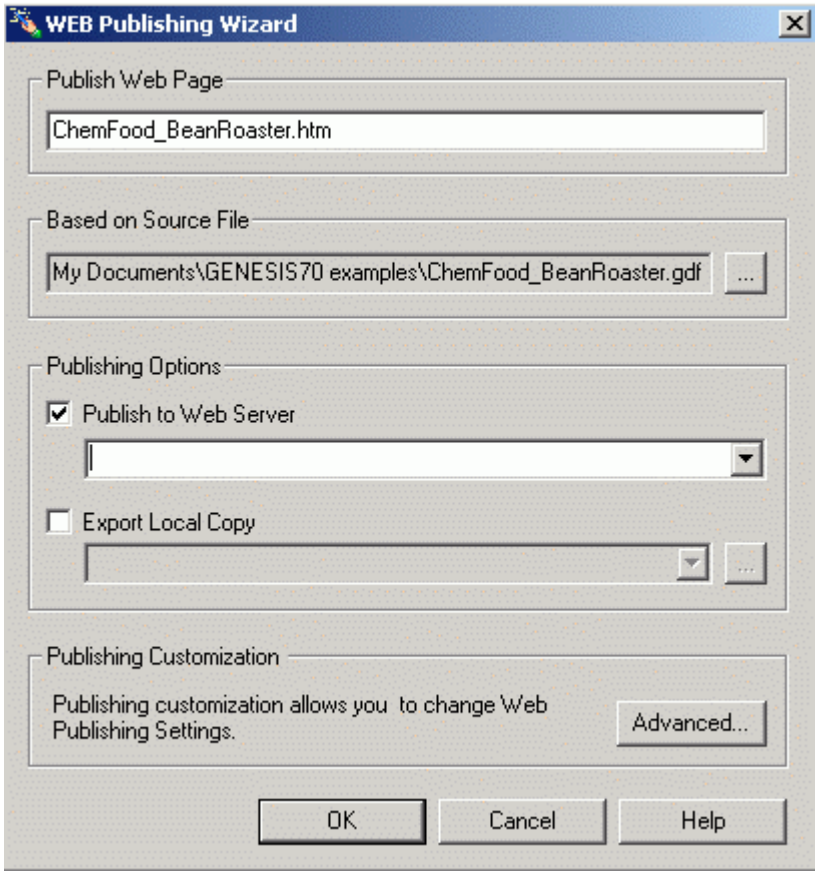
5. Click the **OK** button to generate the HTML file. The new HTML file is saved to the local directory pathway specified in the **Export Local Copy** field.

Note: Both the **Publish to Web Server** and **Export Local Copy** actions can be performed at the same time.

Publishing a Display File to a Web Server

To publish the HTML file to a directory on a Web server (i.e. over the Internet or an intranet):

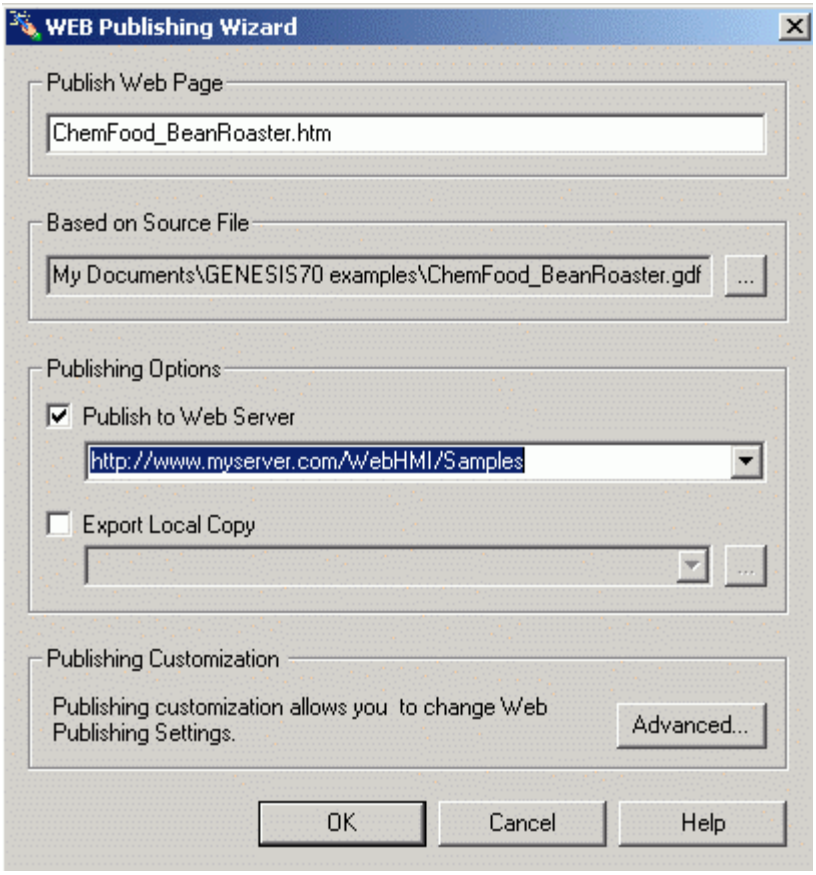
1. In the **Based on Source File** field of the Web Publishing Wizard, you must specify the name of the GraphWorX32 display (.gdf) file to be published, as shown in the figure below. To choose a different display, click the **…** button to the right to browse for a file. Select a file and then click **Open.** The directory path and the file name are shown in the text field.



Specifying a GraphWorX32 Display File

2. In the **Publish Web Page** field of the Web Publishing Wizard, specify the name of the HTML file that will be created. The .gdf file name is filled in by default, but you can give the HTML file a different name.
3. In the **Publishing Options** field, check the **Publish to Web Server** check box and type in the URL address of your Web server with the complete directory indicating where you want to publish the HTML file, as shown in the figure below. In the sample Web server URL address shown below ("http://www.myserver.com/WebHMI/Samples"), the various components are:
    - **http://www.myserver.com:** IP address (server name) of the WebHMI Server
    - **WebHMI:** Name of the WebHMI server root directory
    - **Samples:** Name of the directory on the server to which the HTML file will be saved

4. The Web server URL address now appears in the Web Publishing Wizard dialog box in the **Publish to Web Server** field of the Web Publishing Wizard, as shown in the figure below.



Publish to Web Server Location Specified in Web Publishing Wizard

5. Click the **OK** button to generate the HTML file. The new HTML file is uploaded to the Web server and then saved to the URL address path specified in the **Publish to Web Server** field.

Note: Both the **Publish to Web Server** and **Export Local Copy** actions can be performed at the same time.

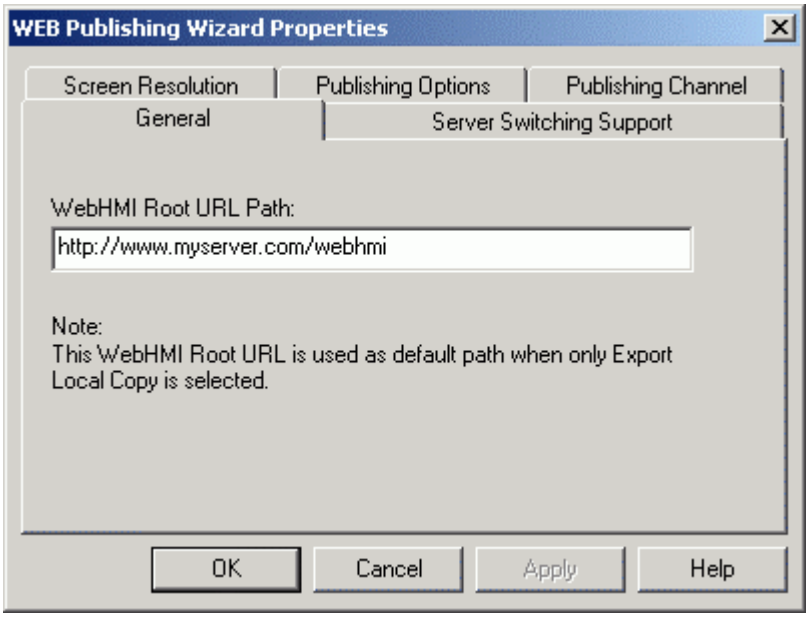
Publishing Customization Options

The Web Publishing Wizard contains some customization options for exporting/publishing HTML files. Clicking the **Advanced** button on the Web Publishing Wizard dialog opens the **Web Publishing Properties** dialog box, as shown in the figure below, which contains the following tabs:

- General
- Server Switching Support
- Screen Resolution Settings
- Publishing Options
- Publishing Channel

General Settings

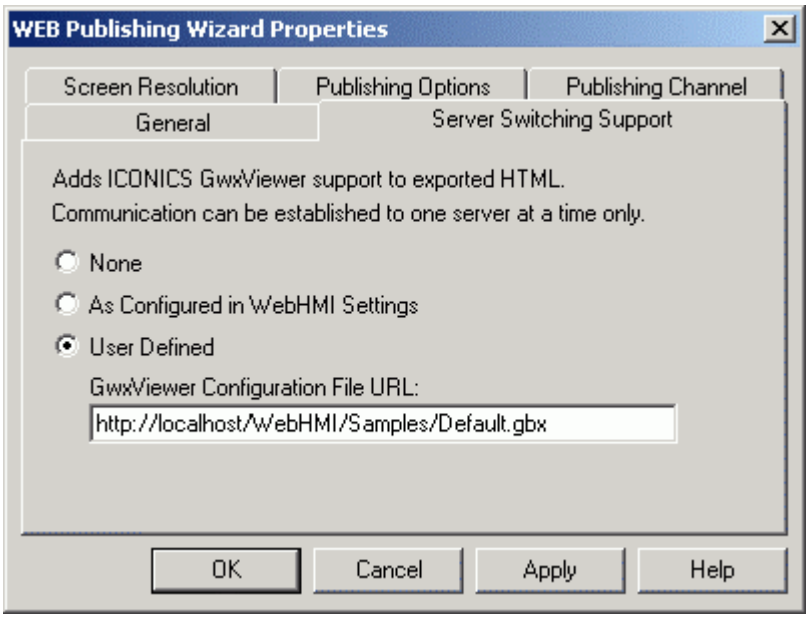
The **General** tab of the **Web Publishing Properties** dialog box, shown in the figure below, allows you to specify the default URL address for the WebHMI Server. The **General** tab is visible only when **Export Local Copy** is the sole option selected on the Web Publishing Wizard dialog box. If **Publish to Web Server** is selected on the Web Publishing Wizard dialog box, then the **General** tab is hidden and you won't need to configure it.



General Settings

Server Switching Support

The **Server Switching Support** tab of the **Web Publishing Properties** dialog box, shown in the figure below, allows you to enable or disable GenBroker support for the exported/published HTML file. Here you can specify which GenBroker configuration file (.gbx) will be activated. The GenBroker configuration file establishes the settings for OPC data communications between the clients and the Web server.



Server Switching Support Settings

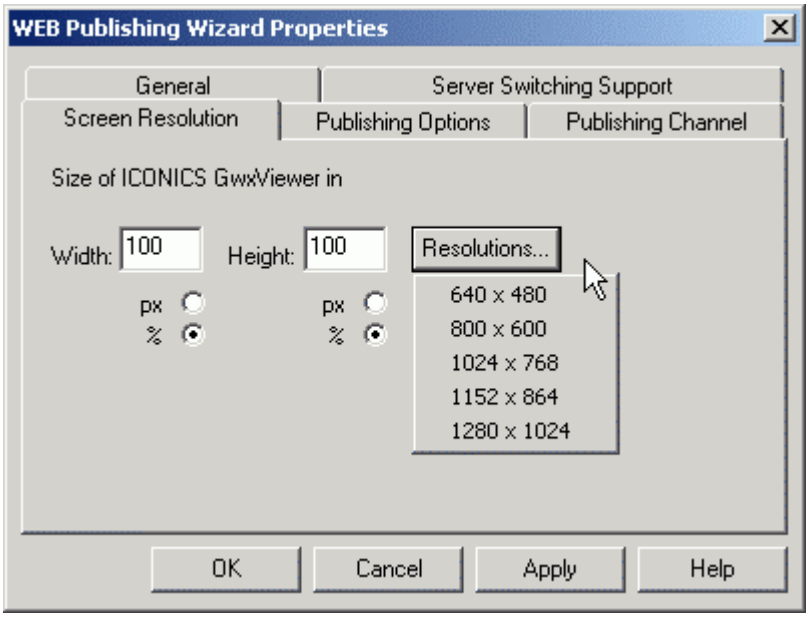
There are three available options for GenBroker support:

- **None:** GenBroker support is disabled. If this option is selected, all the communication will default to the local node. This option should be checked when the Web pages are meant to be used only from the local node. If this option is checked, Web pages may not work on a remote node, unless the remote node has GENESIS32 installed and GenBroker is properly configured to communicate with the WebHMI server.
- **As Configured in WebHMI Settings:** This setting uses the default URL address of the GenBroker configuration (.gbx) file as specified in the WebHMI installation. This is the recommended option for non-expert users.
- **User Defined:** When this option is selected, the **GenBroker Configuration File URL** field becomes available, allowing you to specify the URL address of the GenBroker configuration (.gbx) file. This option gives the maximum flexibility and should be used by expert users only.

Note: For information about configuring GenBroker settings, please see the GenBroker Configurator Help documentation.

Screen Resolution Settings

The **Screen Resolution** tab of the **Web Publishing Properties** dialog box, shown in the figure below, determines the screen resolution and size of the GraphWorX32 Viewer ActiveX control, which is referenced in the generated HTML file and then downloaded to a client PC when the HTML file is viewed in the client's Web browser. You can specify the screen size in the **Width** and **Height** fields in terms of pixels (px) or percentage (%). Click the **Resolutions** button to select from a pop-up menu of standard screen resolutions, as shown in the figure below.

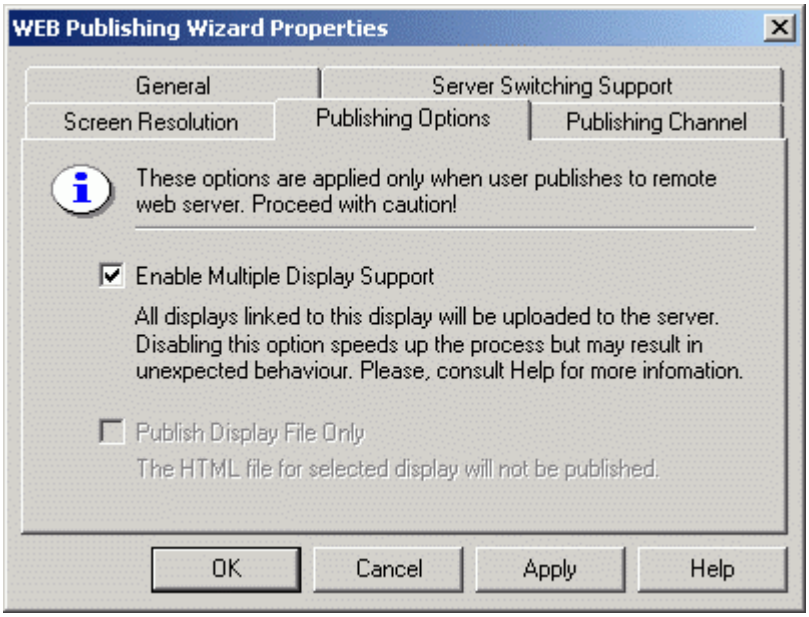


Screen Resolution Settings

Publishing Options

The **Publishing Options** tab of the **Web Publishing Properties** dialog box, shown in the figure below, enhances the publishing process. By default, the Web Publishing Wizard provides support for related files detection (for more details, see the **Multiple Display Support** section). The multiple display detection could be a rather lengthy process. You can speed up the process by disabling the **Enable Multiple Display Support** option. This is especially useful, if you have already published your project files to the Web server, and now you want to update display you have changed.

The multiple display detection mechanism ensures that the Web page will be published properly for the source file. Disabling the **Multiple Display Support** may cause publishing of an incomplete web page, which may result in unexpected behavior. Therefore, it is suggested to disallow publishing of the Web page. You can do so by enabling the **Publish Display File Only** option.



Publishing Options

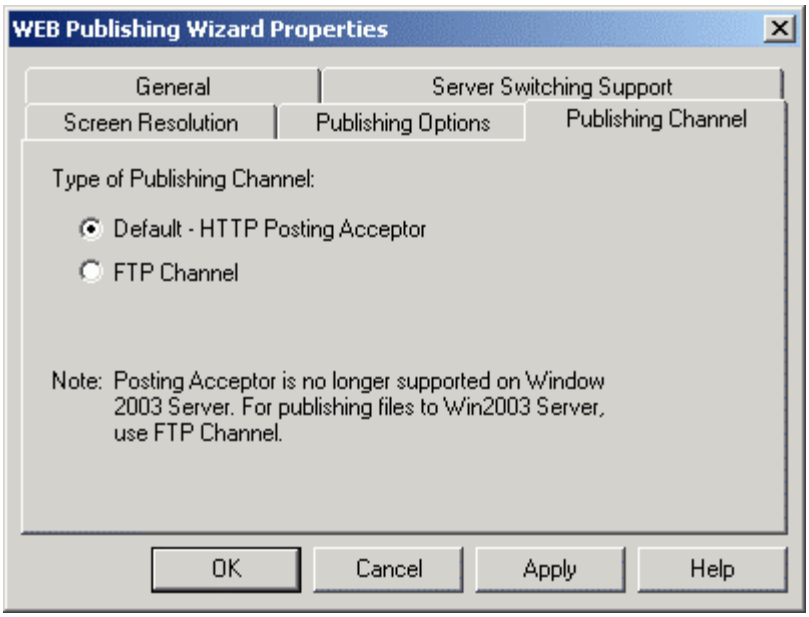
Publishing Channel

Because publishing via Microsoft Posting Acceptor is no longer supported on Windows Server 2003, we were forced to find another way of transporting published files from a GENESIS32 development machine to a WebHMI server. It appeared that FTP would be the best choice.

Changing the Communication Channel

By default, the method that uses Posting Acceptor services is used for publishing files to a Web server. The **Publishing Channel** tab of the **Web Publishing Properties** dialog box, shown in the figure below, enables you to change the way that files are uploaded to the Web server by selecting **FTP Channel.** This tells the Web Publishing Wizard to use FTP communications for uploading files instead of Posting Acceptor.

Note: Selecting **FTP Channel** does not affect the configuration of the main Web Publishing Wizard dialog; the settings are the same if you were publishing files using the default method (i.e. Posting Acceptor).

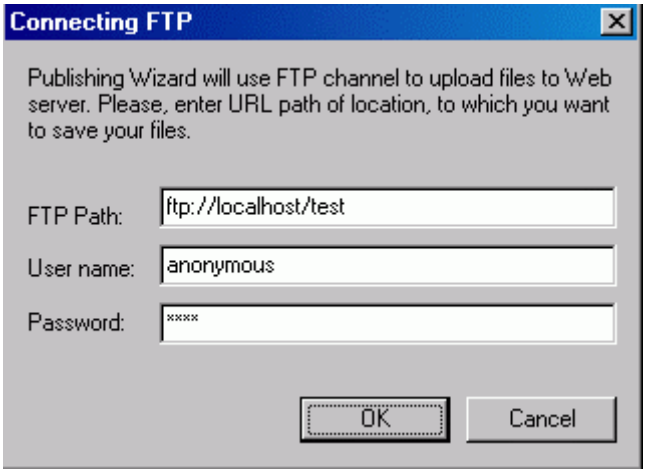


Publishing Channel

Configuring FTP Access

Just before uploading files to the Web server, you will be prompted to specify the FTP connection settings, as shown in the figure below. Enter the correct FTP path, user name and password.

Note: For information about FTP site configuration, please see the application note called “Publishing to Windows 2003 Server,” which is located on the GENESIS32 CD.



Uploading Files Via FTP

Viewing Locally Exported HTML Files

To view an HTML file that was exported to a local directory, simply browse to the local directory pathway and click on the file to open it in your Web browser. You can also open your Web browser and copy the directory pathway of the file into the **Address** field of the Web browser, as shown in the figure below. A complete directory path is, for example, "C:\Documents and Settings\Administrator\My Documents\Web Publishing Example\ChemFood_BeanRoaster.htm." Be sure to include the name of the HTML file that you have exported and the file extension "htm."

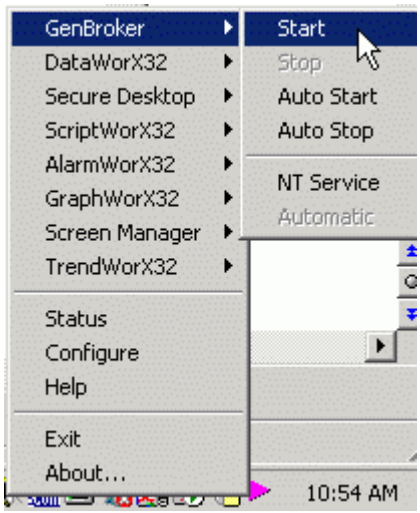


Viewing the Exported HTML File in a Web Browser

Viewing Published HTML Files

Before connecting a client machine to a WebHMI Server, you must start the GenBroker Server on the WebHMI Server using GenTray. If the GenBroker Server is not started, then the client will not get proper licensing and security information. To start the GenBroker Server on the WebHMI Server:

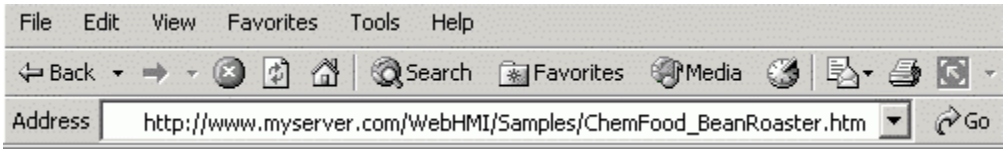
1. Launch GenTray from the Windows **Start** menu by selecting **Programs > ICONICS GENESIS32 > Tools > GenTray.**
2. When GenTray opens, the triangle icon will appear in the Windows tool tray. Click on the triangle and select **GenBroker > Start** from the pop-up menu, as shown in the figure below. This activates the GenBroker Server.



Starting the GenBroker Server on the Server Node

3. Once the GenBroker Server is running, you can view the HTML file that you have published to the Web server from any client PC's Web browser. To view the published HTML file, open your Web browser and copy the URL address of the file into the **Address** field of the Web browser, as shown in the figure below. A complete address is, for example, "http://www.myserver.com/webhmi/Samples/ChemFood_BeanRoaster.htm." Be sure to include the name of the HTML file that you have published and the "htm" file extension.

The GraphWorX32 display file is referenced in the HTML code so the display can be viewed as a Web page. The client's Web browser simply downloads the HTML file in which the .gdf file is referenced. All the client needs is a Web browser; it is not necessary to have GENESIS32 installed on client. The Web page is downloaded from the Web server across the Internet/intranet and appears in the client's Web browser window. The display is in real-time, just as if you were viewing the runtime display in GraphWorX32 on the server machine; any live values (e.g. OPC tags) change dynamically in the display.



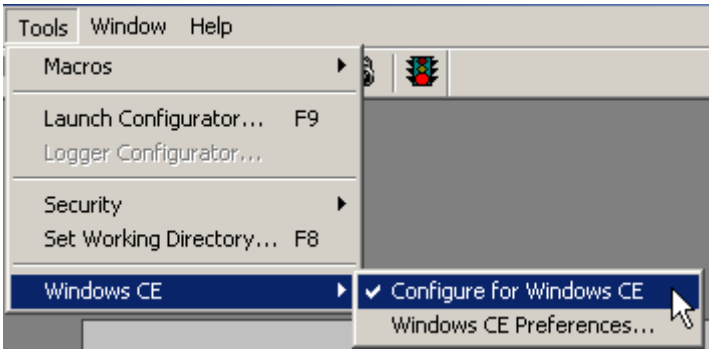
Viewing the Published HTML File in a Client Web Browser

Download to Windows CE Support

All display files for Pocket GENESIS are first configured on a desktop PC using a GENESIS32 application (such as GraphWorX32, TrendWorX32, and AlarmWorX32) and then saved in a format that is compatible with Pocket PC and Windows CE. A display file is then downloaded to the Pocket PC or Windows CE device to be viewed during runtime. Once you open the display file in the corresponding Pocket GENESIS application (i.e. Pocket GraphWorX, Pocket TrendWorX, or Pocket AlarmWorX) in runtime mode, you can execute all runtime functional dynamics that have been added during configuration. With a few exceptions, Pocket GENESIS applications have the same basic runtime functionality as their counterpart GENESIS32 applications.

Configuring for Windows CE in GENESIS32

To configure for Windows CE in GraphWorX32, TrendWorX32, and AlarmWorX32, select **Windows CE** > **Configure for Windows CE** from the **Tools** menu, as shown in the figure below. If you have just installed Pocket GENESIS, this will be checked by default.



Configuring for Windows CE

Once you have configured your display, select **Save As** from the **File** menu. This opens the **Save As** dialog box, which allows you to save your configuration as a display for Windows CE file. Give the file a name and select one of the following file types from the **Save As Type** box:

- In GraphWorX32, select either "GraphWorX32 Displays for WinCE (*.gdc)" or "GraphWorX32 Templates for WinCE (*.gdc)."
- In AlarmWorX32, select "AWXview32 CE File (*.ace)."
- In TrendWorX32, select "TWXviewerCE File (*.tce)."

Click the **Save** button. Upon saving your configuration file for Windows CE, features that are not supported by Pocket PC or Windows CE will be removed from the Pocket GENESIS display.

Note: When opening a Pocket GENESIS display in GENESIS32, you must specify the proper Windows CE file extension.

Remote Connections

In order to connect remotely to OPC servers over a network, you must install and use GenBroker. Please see the GenBroker Help documentation for more information about configuring GenBroker.

Downloading GENESIS32 Configuration Files to Your Pocket PC

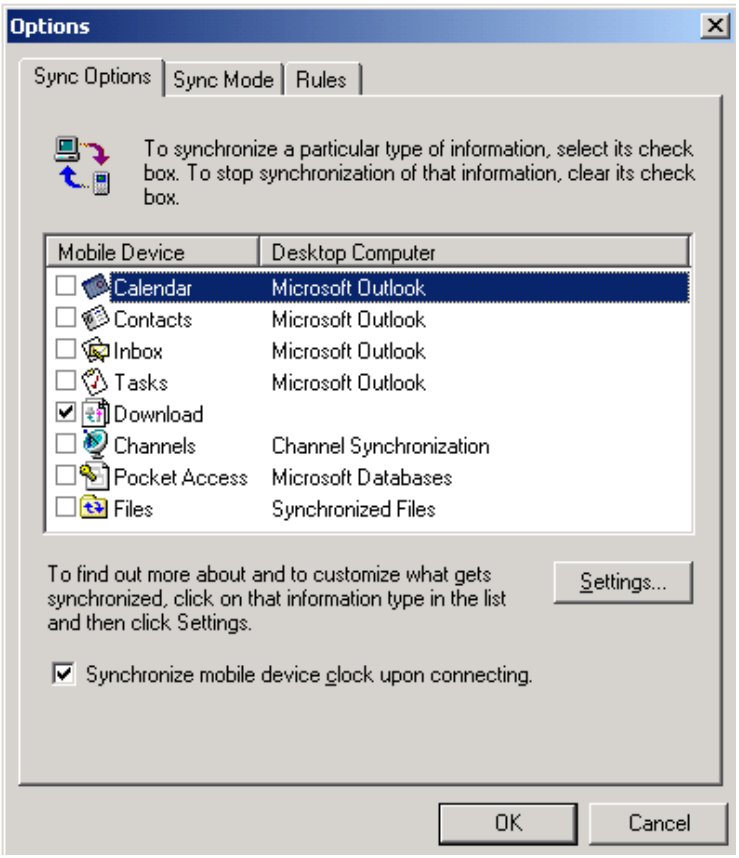
Once you have configured your application in GENESIS32 and saved the Pocket GENESIS display file, you can use the file Download to Windows CE Utility in GENESIS32 to download configuration files from GraphWorX32, TrendWorX32, and AlarmWorX32 to your Pocket or CE device. When developing a configuration file for a Pocket GENESIS application, this feature allows you to download the configuration file to a Windows CE or Pocket PC device. The file download function uses Microsoft ActiveSync to connect to the CE device.

Setting up the Download

The download to CE tool requires modules on both a desktop or workstation PC as well as a Pocket PC or CE device, because Microsoft ActiveSync services are used for connecting and authenticating the CE device.

Configuring the Desktop

The desktop is the only part that has to be configured. If the file download tool is installed properly, it will be listed in the ActiveSync Manager list of ActiveSync modules in the **Sync Options** tab of the Windows **Options** dialog box. You must enable the file download ActiveSync module in the ActiveSync Manager by checking the **Download** box, as shown in the figure below.

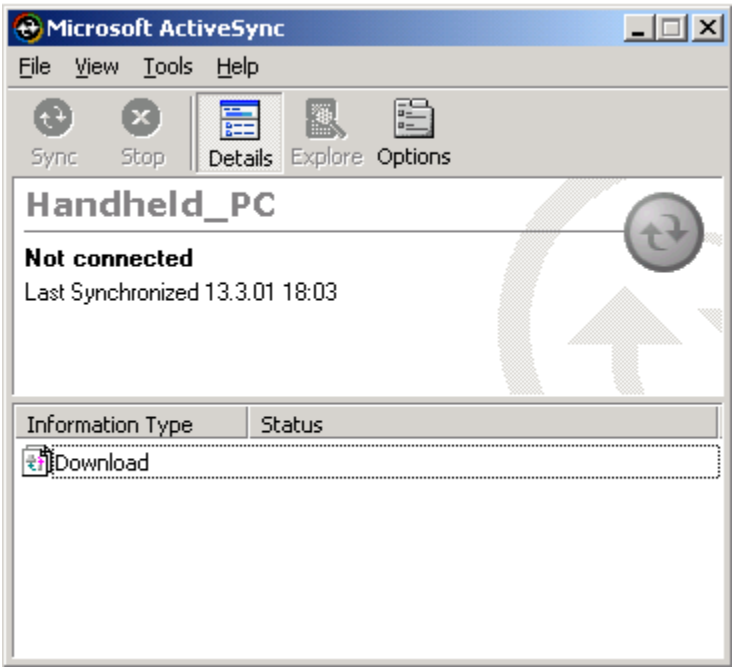


Windows Options Dialog Box

An association with the CE device must be created to synchronize the download ActiveSync modules, as shown in the figure below. When the CE device responds without any problems, the download synchronization module is active.

When enabled, the file download tool immediately downloads the configuration file for Windows CE to the CE device. When disabled, the configured file will only be saved.

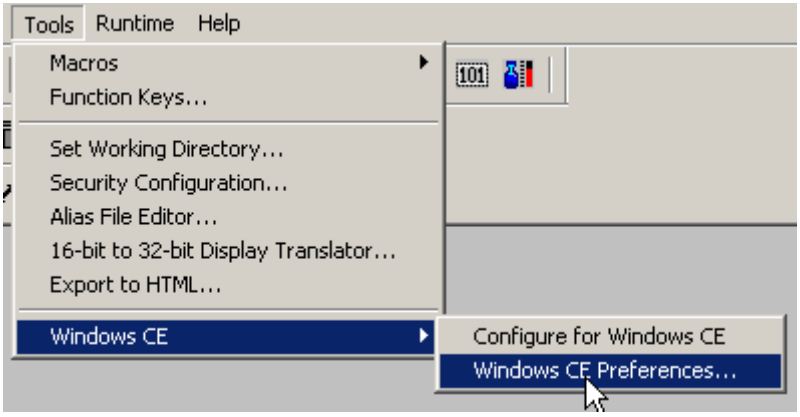
For more information about Microsoft ActiveSync services, please refer to the Microsoft ActiveSync help documentation.



Microsoft ActiveSync

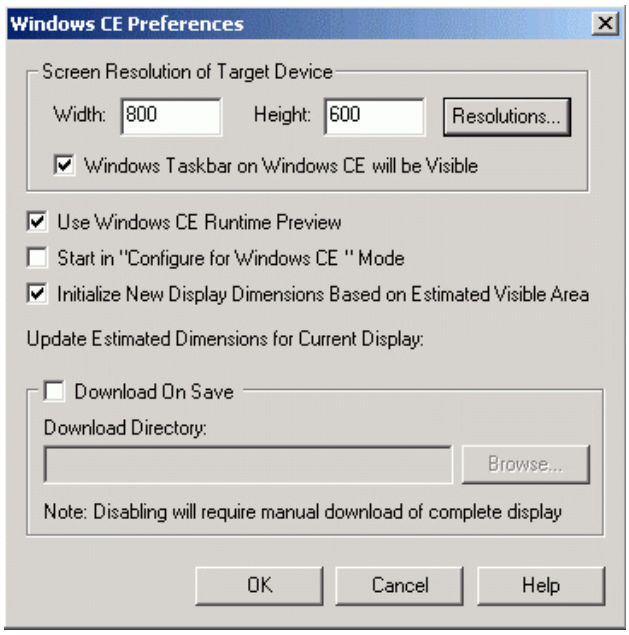
File Download Application Configuration

GraphWorX32, TrendWorX32, and AlarmWorX32 have their own version of the file download tool. In these applications, choose **Tools > Windows CE > Windows CE Preferences,** as shown in the figure below.



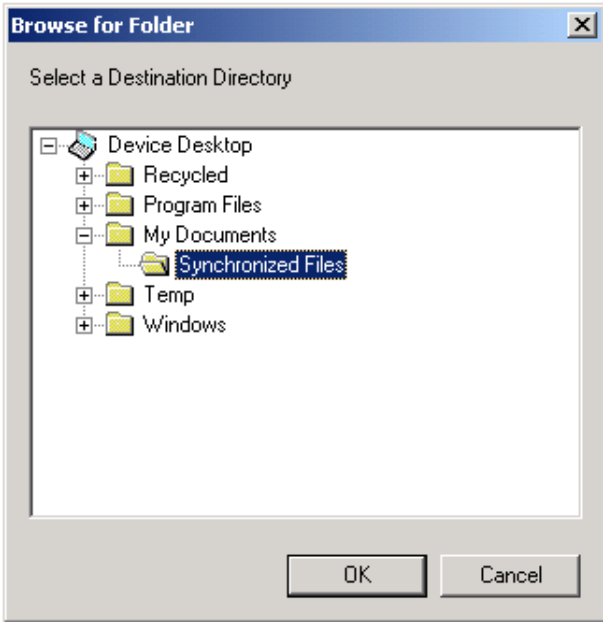
Starting the Windows CE Download Tool

This opens the **Windows CE Preferences** dialog box, which allows you to enable or disable the file download tool. To enable this tool, check the **Download On Save** check box. Then select the destination directory on the CE device by entering the path name in the **Download Directory** field, as shown in the figure below.



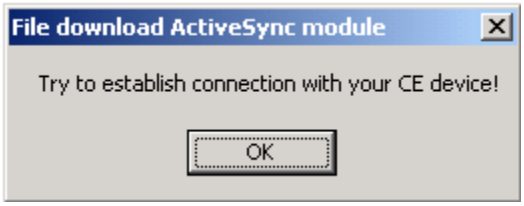
Setting Windows CE Preferences

You can also click the **Browse** button to select the destination directory. Clicking the **Browse** button opens the **Browse for Folder** dialog box, which shows the CE device directories, as shown in the figure below. A warning message box will be displayed when the file download ActiveSync module is inactive.



Browsing for a Folder on the CE Device

To establish a connection with the CE device, click **OK,** as show in the figure below.



File Download ActiveSync Module

You can also use the **Windows CE Preferences** dialog box to configure other settings for the CE device. You can set the resolution of the target device by clicking the **Resolution** button. This displays a list of resolutions from 200 x 320 up to 1024 x 768.

Downloading the File to the CE Device

When you have finished configuring a display file and would like to save it to Windows CE, choose **Save As** from the **File** menu. This opens the **Save As** dialog box.

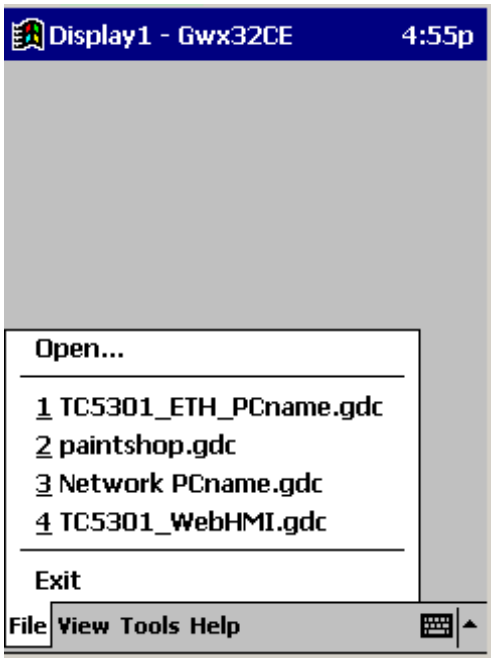
- In GraphWorX32, enter the file name and select either "GraphWorX32 Displays for WinCE (*.gdc)" or "GraphWorX32 Templates for WinCE (*.gdc)" from the **Save As Type** box.
- In AlarmWorX32, enter the file name and select "AWXview32 CE File (*.ace)" from the **Save As Type** box.
- In TrendWorX32, enter the file name and select "TWXviewerCE File (*.tce)" from the **Save As Type** box.

The ICONICS **Import File** dialog box will be displayed, indicating that the files are downloading to the Windows CE device.

Starting Pocket GENESIS Applications on Your Pocket PC

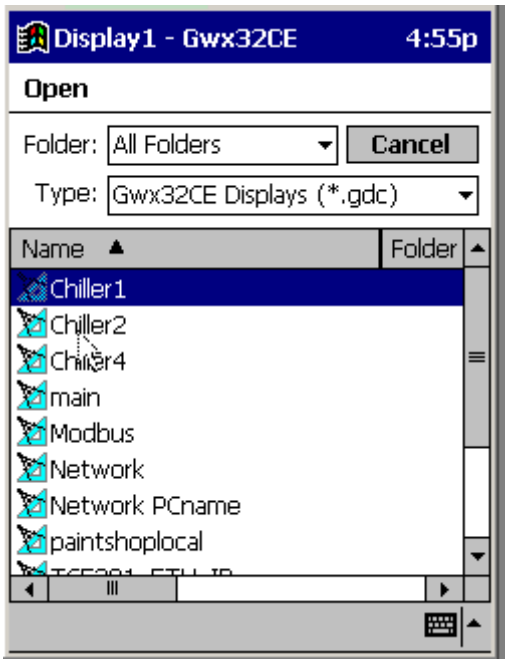
Once you have downloaded your GraphWorX32, AlarmWorX32, or TrendWorX32 configuration files from your desktop PC to your Pocket PC or CE device, you can run the applications on your Pocket PC. Open the Pocket GENESIS client applications (Pocket GraphWorX, Pocket AlarmWorX, and Pocket TrendWorX) you wish to run, and then select **Open** from the **File** menu. Browse for the desired .gdc, .ace, or .tce file. An example using Pocket GraphWorX is provided below.

1. Open Pocket GraphWorX on your Pocket PC. In Pocket GraphWorX, select **Open** from the **File** menu, as shown in the figure below.



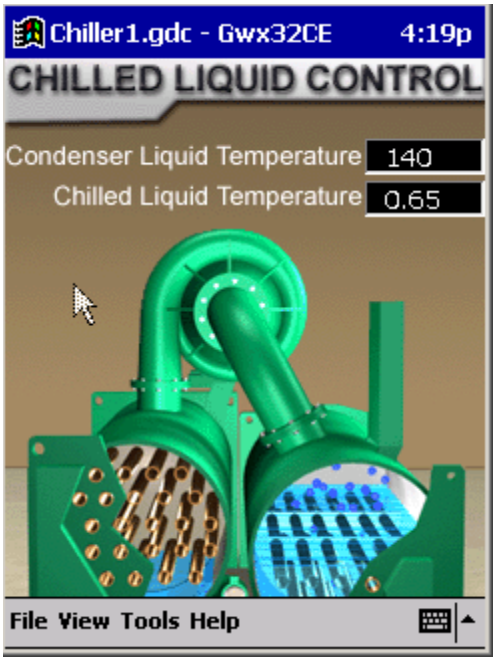
Opening a Pocket GraphWorX Configuration File

2. Browse for a Pocket GraphWorX display (.gdc) file, as shown in the figure below. In this example, the **chiller1.gdc** file is selected.



Browsing for a Display File

3. The display file opens in Pocket GraphWorX, as shown in the figure below. The runtime operations of Pocket GENESIS enable you to see real-time OPC data values in the display.



Viewing the Display in Pocket GraphWorX

Note: The procedure for opening and viewing files in Pocket AlarmWorX and Pocket TrendWorX is similar to the procedure described above for Pocket GraphWorX. For detailed information about each Pocket GENESIS application, please see the Help documentation for that application on the Pocket GENESIS CD.

Scripting Engine and Script Editor for VBScript and JScript

GraphWorX32 has a Visual Basic for Applications (VBA) integrated editor, which allows writing of VBA Script. This is a powerful tool that allows easy expansion of GraphWorX32 functionality. In order to run a VBA Script, a PC must have the VBA Runtime Engine. This engine cannot be delivered through the Internet on a thin-client machine because it takes a lot of disk space and requires a long download time. Because of this, in the past ICONICS has decided to remove VBA support from WebHMI and the other “lightweight” products, such as Pocket GraphWorX. So it is impossible to use the power of VBA and the advantage of the thin-client technology of WebHMI. Many users develop two different sets of displays: one set with VBA for the desktop workstation, and another set without VBA for the WebHMI thin client. This requires additional work needed to strip out the VBA code and redesign the display.

As a solution to the problem described above, GraphWorX32 features a scripting engine that can parse VBScript and Jscript code. This engine is lightweight and can easily be distributed over the Internet to any thin-client machine. When you plan to use the scripting technology for enhancing GraphWorX32 features and also plan to use thin-client technology, such as WebHMI, you should use the new embedded scripting engine and write your code either in VBScript or Jscript instead of Visual Basic for Applications. You can now design one unique display with script and WebHMI support, without having to manage multiple displays.

The scripting engine for VBScript and Jscript does not have all the power of a full VBA engine, so some features are not supported. There are many limitations, but the largest one is probably the fact that it is impossible to handle events coming from ActiveX or other controls embedded inside a GraphWorX32 display.

The new scripting engine is able to handle the most important display events, and all the pick actions, but it is currently impossible to fire new events.

An additional limitation is that VBScript and JScript do not support forms, so it is impossible to create user interface elements different from a simple message box or input box.

Note: 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:

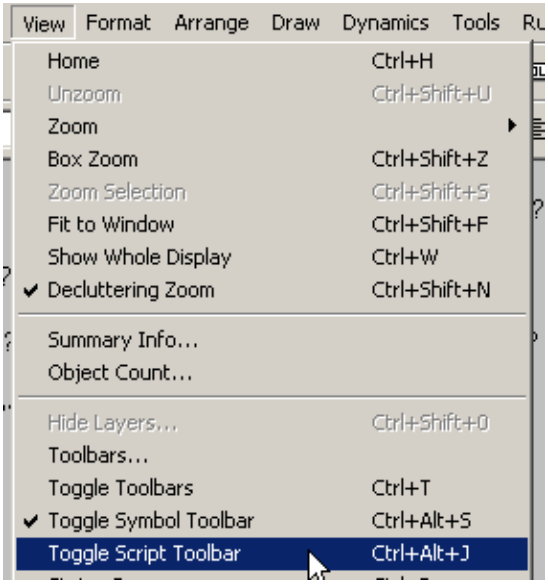
```
ThisDisplay.FileOpenSetAliases("myFileName.gdf", 1,
"AliasCommandString.txt")
```

Displaying the Script Editor

The integrated Script Editor appears as a toolbar in the GraphWorX32 display. It can be dragged around the display as a floating window or it can be docked in place on one of the four display sides. You can resize the toolbar (both docked and floating) at any time by dragging its border. The Script Editor is an

integrated toolbar, so you do not need to “open” it. Instead you just “show” or “hide” it by selecting **Toggle Script Toolbar** from the **View** menu, as shown in the figure below.

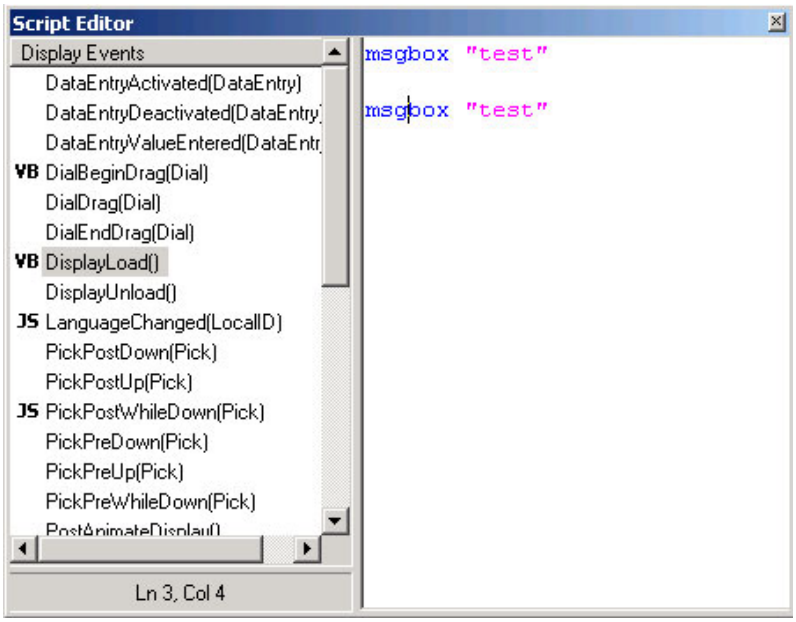
Note: You can also use the convenient shortcut key combination **CTRL+ALT+J,** or select **Toolbars** from the **View** menu and check the **Script** check box on the **Show Toolbars** dialog box. In addition, you can toggle the Script Editor toolbar by clicking the "pencil" button on the **Main** toolbar.



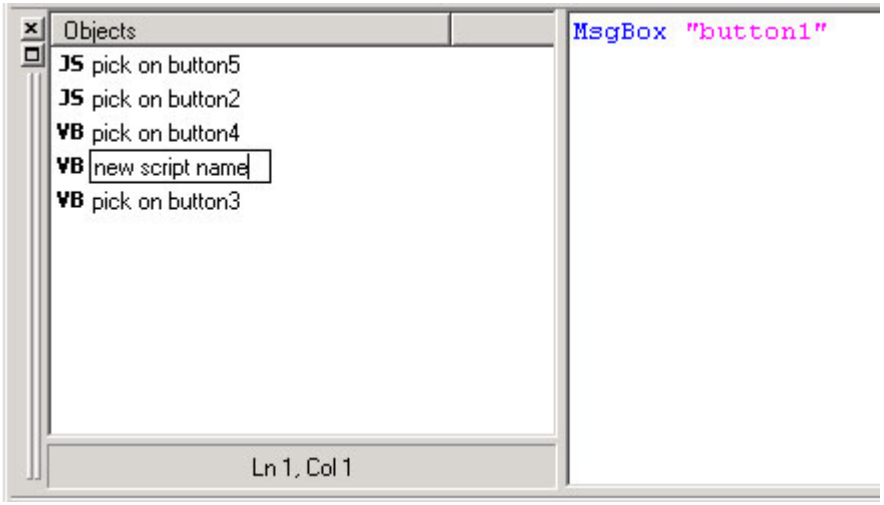
Starting the VBScript / Jscript Events Editor

The **Script Editor** is displayed, as shown in the figure below. The editor is divided into a left panel and a right panel. The left panel is the script browser, and the right panel shows the code for the selected script. Each script shown in the left panel browser is identified as VBScript **(VB)** or JScript **(JS),** as shown in the figure below. You can change the size of the left and right panel of the script by dragging the splitter bar between them.

The Script Editor has two main editing “modes”: **event mode** and **object mode.** When the Script Editor is in event mode, the left panel lists all the display events that can be scripted. When the Script Editor is in object mode, left panel lists the scripted objects in the display. You can switch the script editor mode by clicking on the left panel header. The figures below show the event and object modes.



Script Editor in Display Events Mode



Script Editor in Objects Mode

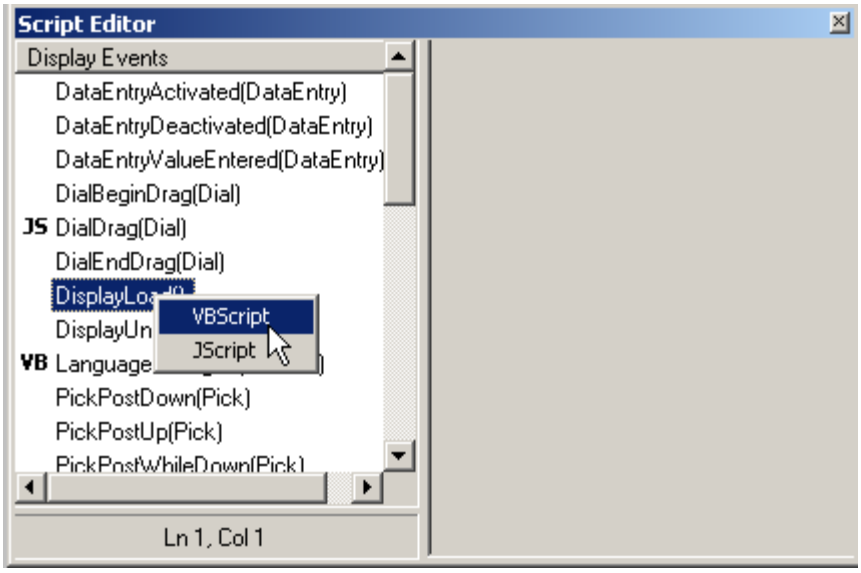
Each script and its corresponding object is synchronized in the display. Therefore, if you select an object in the display that has a script associated with it, then the script is automatically highlighted in the Script Editor. Conversely, you select a script in the Script Editor, then the corresponding object is also selected in the display.

Event Mode

When the script editor is in **event mode,** then the left panel lists all the display events that can be scripted. For each event, it is possible to write a script to handle it.

A complete list of these events can be found in the GraphWorX documentation.

To associate a script with an event, right-click the event in the script browser (left panel). A pop-up menu appears, as shown in the figure below. You can then choose the language that you want to use to handle the script (VBScript or JScript).

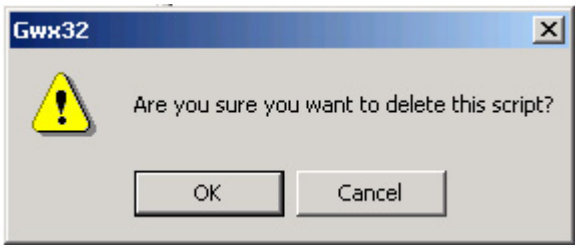


Selecting the Script Type

Once you have chosen the script type then you can start editing the script.

The next time you open the pop-up the menu, the script type is indicated by a check mark. You can change the script type at any time.

Note: If you uncheck the item, you will delete the script. A dialog will ask to confirm before deleting the script, as shown in the figure below.



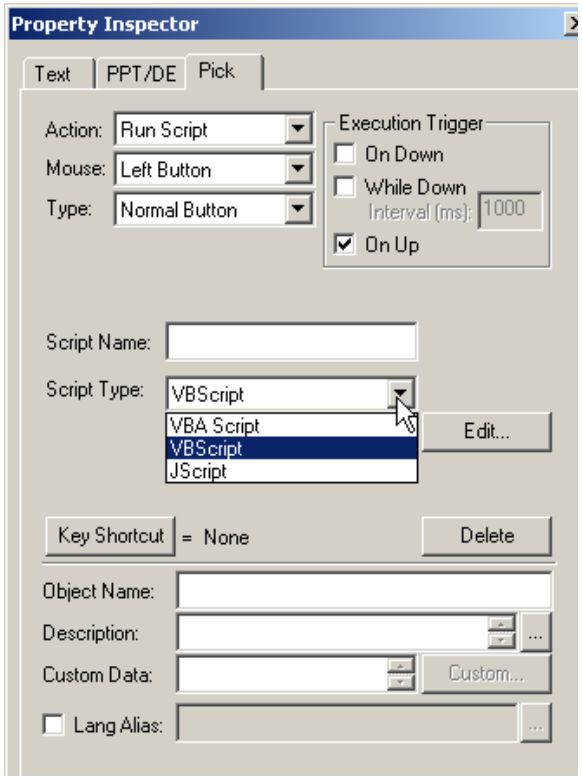
Deleting a Script

Object Mode

When the Script Editor is in object mode, you can edit the script associated with an object. It is possible to associate a script with an object trough the “pick action” dynamic. A pick action is a special kind of dynamic that is executed in each time you click on the object. When the script editor is in object mode, the left panel lists the scripted objects in the display. A **scripted object** is any GraphWorX object that has a “run script” pick action associated with it and has the type “VBScript” or “Jscript.” The Script Editor never shows the objects with the regular VBA script pick action. The VBA script can be edited only with the standard VBA Editor.

To create a new script and associate it to an object you have to add a “pick action” dynamic to the object:

1. In the **Pick** tab of the Property Inspector, choose the **Run Script** pick action and type a name for the script in the **Script Name** field. If you omit the script name, GraphWorX32 will automatically use the object name property of the pick object. (If it also is omitted, then GraphWorX32 will automatically associate a name with this script.
2. Select the type from the **Script Type** drop-down list, as shown in the figure below. Only VBScript and JScript can be edited with the Script Editor.



Choosing a Script Type for the Run Script Pick Action

3. Click **OK.** When you choose VBScript or JScript, a new script is generated in the **Script Editor.** The script name automatically appears into the Script Editor. If the Script Editor was in event mode, then it automatically switches to object mode and selects the script that you have just created. Can then start to write the script.

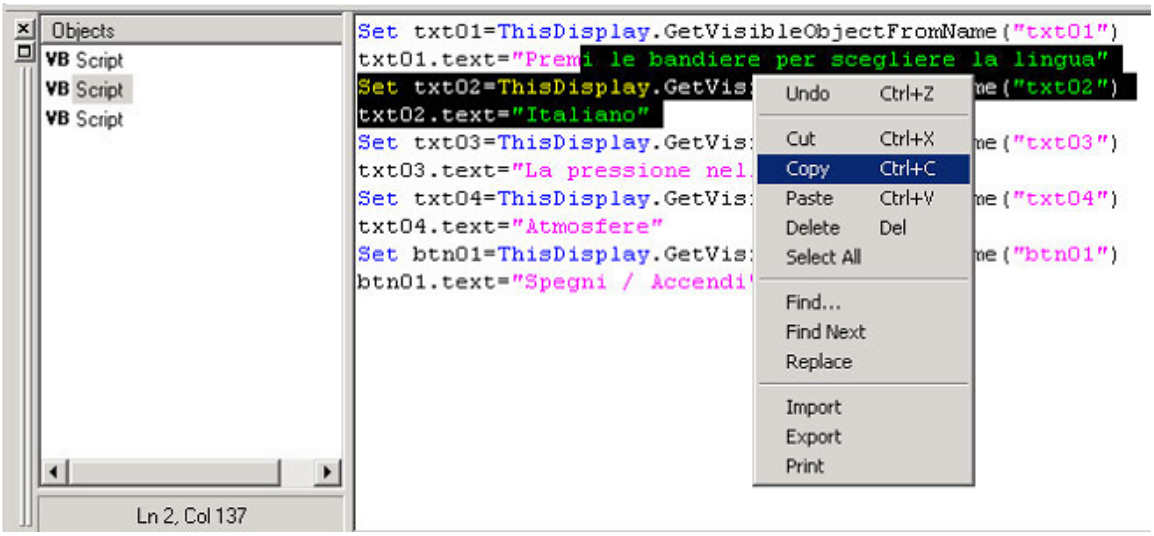
Changing the Script Name

You can change the name of a script at any time either through the object property page or directly in the Script Editor. When the editor is in object mode, you can rename the currently selected script by clicking on the script name inside the script browser (left panel).

Note: It is not possible to change the script name when the editor is in event mode. In other words, you cannot change the default name for the events.

Basic Text Operation

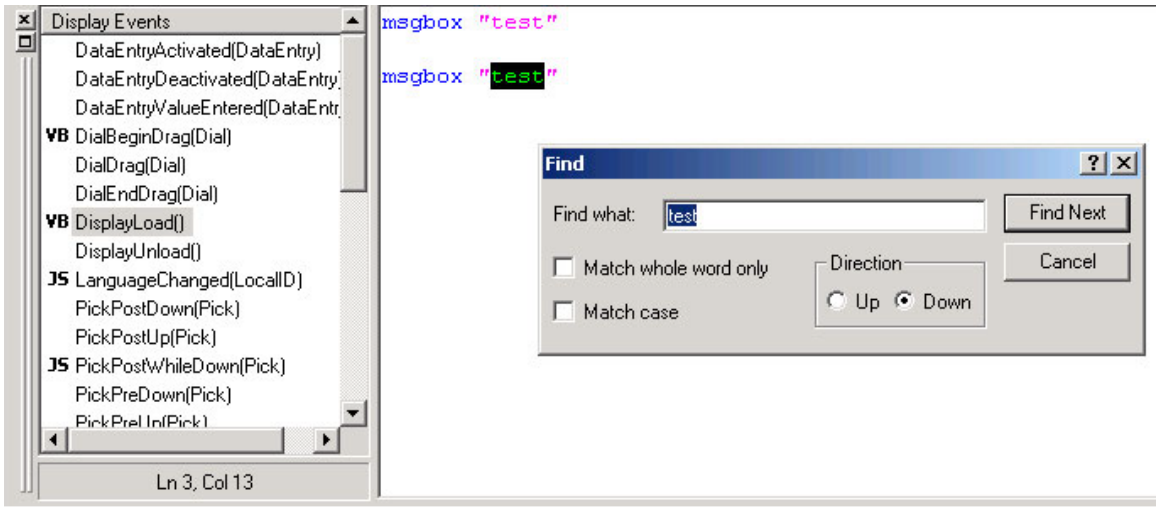
When you right-click on the script (right) panel, a pop-up menu appears, as shown in the figure below. This menu contains all the basic text operations: **Undo, Cut, Copy, Paste, Delete,** and **Select All.**



Editing Text in the Script Editor

Text Search and Replace

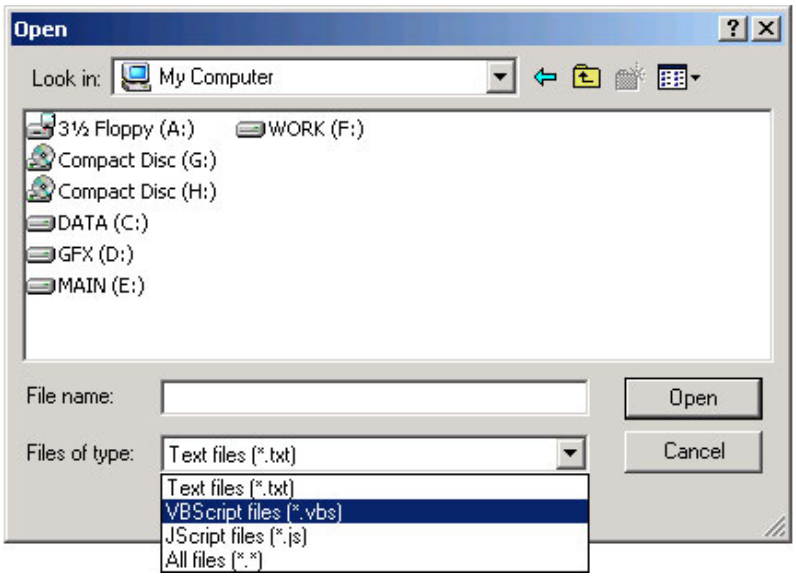
Right-clicking the script (right) panel and selecting **Find, Find Next,** or **Replace** allows you to do text search and replace functions, as shown in the figure below. You can choose few advanced options, which allow you to match the whole word only and to match the case. You can also specify whether to search down or up and to replace only the first instance of the searched text or all of the instances present in the script text. You can keep searching in the text by clicking **Find Next** on the pop-up menu.



Basic Search and Replace

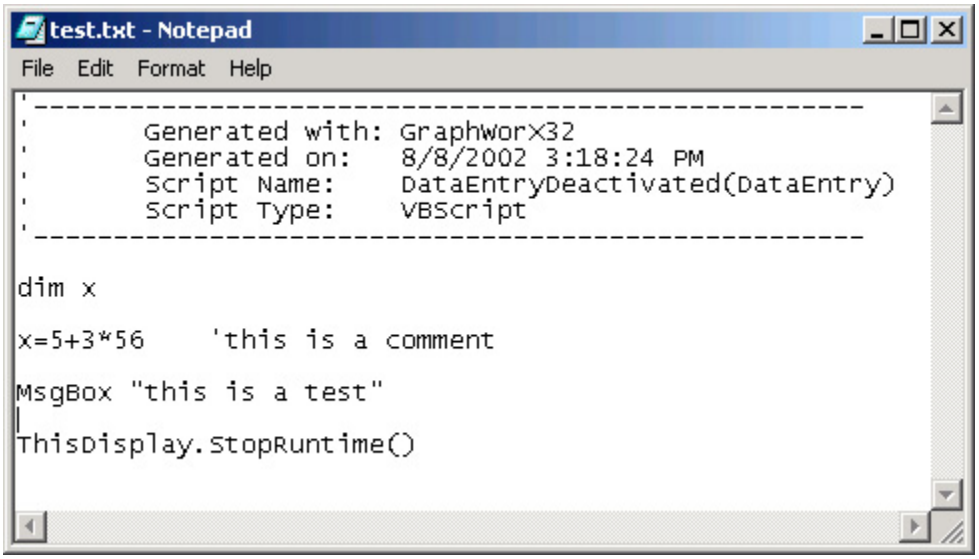
Importing and Exporting Scripts

When you right-click the script (right) panel and select **Import** or **Export,** you can import and export the script text from or to a separate file. You can import text into the script editor from any text file. You can also import text from the standard VBScript and JScript files (*.vbs and *.js), as shown in the figure below.



Importing and Exporting Scripts

You can export your current script to a text file, a VBScript file or a JScript file. Each time you export a script, GraphWorX32 automatically adds a header that describes the file content and the time it was generated, as shown in the figure below. Notice the application header with the timestamp, the script type, and the script name. In this case, the script was associated with an event, so the header has the event name as a script name.



Script Exported to Text File

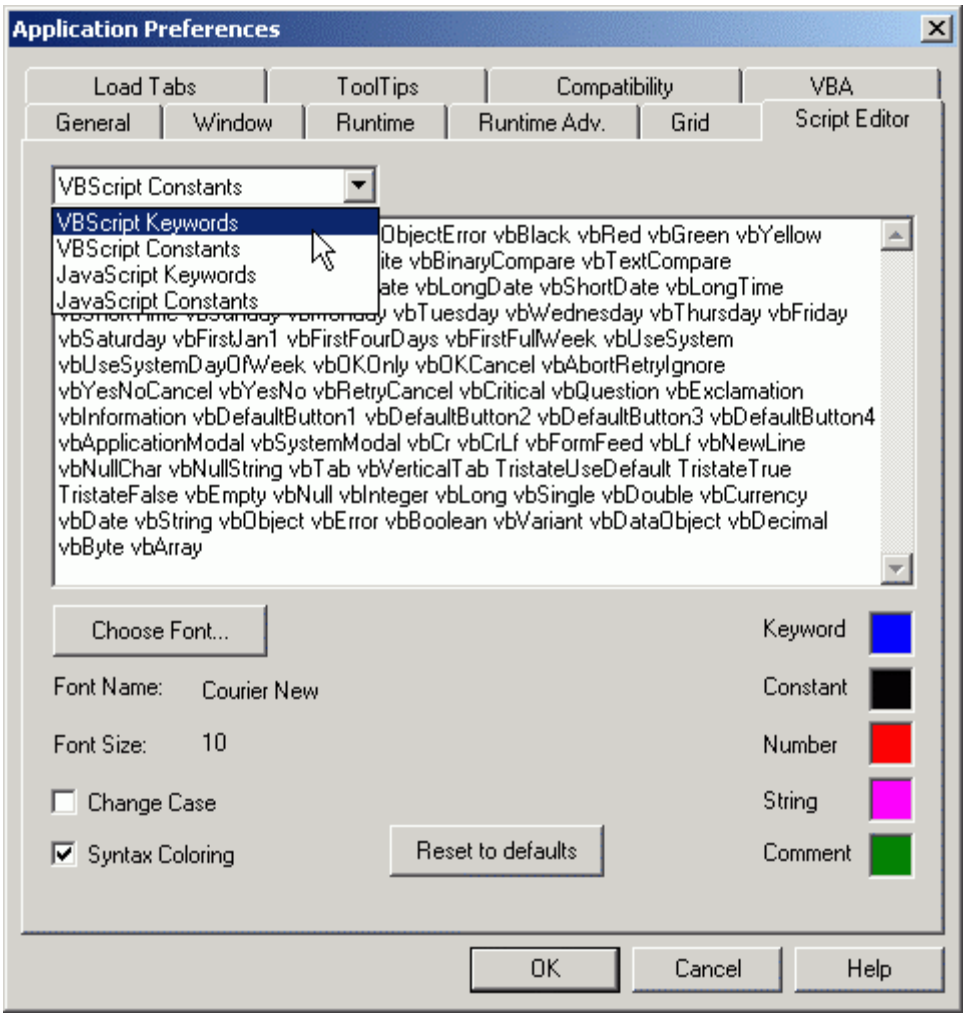
Print Scripts

You can print script directly from the script editor by right-clicking the script (right) panel and selecting **Print** from the pop-up menu. This opens the standard Windows Print dialog box where you can configure the printer settings. You can choose the printer that you wish to use, and you can also choose the number of copies that you want to print. There is no support for printing only a range of page or only the selected text. The support for “collate” is also disabled.

Preferences and Settings

The Script Editor is designed to be customizable for users with special needs. The customization settings can be configured in the **Script Editor** tab of the **Application Preferences** dialog box by selecting **Application Preferences** from the **Format** menu. You can change the following Script Editor settings, as shown in the figure below:

- Font face, size and style
- VBScript keywords and constants
- JScript keywords and constants
- Color for keyword, constants, string, number, and comment
- Enable or disable the syntax coloring
- Enable or disable the automatic case change feature



Application Preferences: Script Editor Tab

Chapter 10: Runtime Environment

This chapter discusses the GraphWorX32 runtime environment. The runtime environment lets you view multiple windows of displays on screen. You can view any combination of the GENESIS32 windows applications (i.e. TrendWorX32 and AlarmWorX32 as well as other applications such as Microsoft Excel.)

The runtime environment functions according to how you configure the runtime parameters in GraphWorx32 configuration mode.

Contents

Starting Runtime
Runtime File Menu
File Functions
Runtime View Menu
Runtime Security
ICONICS Login User Menu
Options
Runtime Data Entry
TraceWorX32 Support
OLExpress Compatibility
VCRWorX32 Data Replay

Starting Runtime

Menu Bar Option

To start runtime mode in GraphWorX32, select **Runtime** on the menu bar.

Shortcut on Desktop

1. Right-click on the shortcut for GraphWorX32 to open the **Properties** menu.
2. Select **Shortcut > Target** and type in "runtime" to enter runtime mode from the shortcut.

Command Line Option

1. Select **Run** on the **Start** menu. Type in the path for GWX32.exe.
2. Specify a display and type in the path name for a display in GraphWorX32 ("display.gdf") to launch the runtime mode with a display.

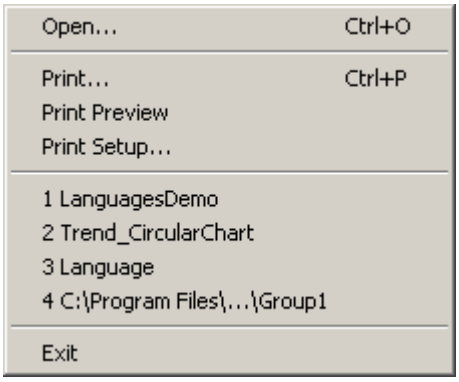
When you enter runtime mode, the menu bar changes to display the following runtime menus:

- File
- View
- Tools
- Configure: Exits runtime mode and returns to configuration mode.
- Help

Runtime File Menu

The File menu commands in runtime mode, shown in the figure below, are described in the sections below:

- File - Open
- Printing the Screen
- Print Preview
- Print Setup



Runtime File Menu

File - Open

To load a display during runtime mode:

1. Select the desired window if you have multiple windows running.
2. Select **Open** from the **File** menu. This opens the **Open File** dialog box.
3. Select a display file and click **OK.** The display appears in the selected window.

Printing the Screen

While in the runtime environment, you can initiate a print screen action to print the entire screen. This function prints the screen to the currently configured Windows printer. To print the full screen:

1. Select **Print** from the **File** menu. This opens the **Print** dialog box.
2. Complete the required fields and click **OK.**
3. The screen is printed immediately.

Print Preview

To preview the display prior to printing:

1. Select **Print Preview** from the **File** menu.
2. The menu bar provides options to preview the current page, previous page, next page, or two pages at once. It also lets you zoom in and out of the display.
3. Click **Close** to exit the preview mode to the full screen mode.

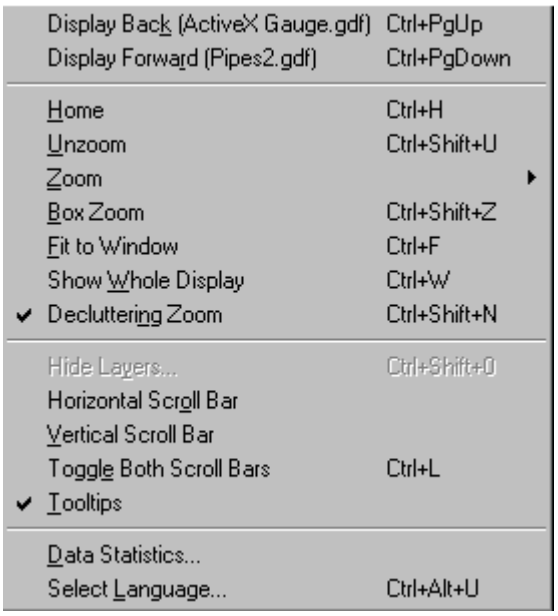
File - Print Setup

To configure printer properties:

1. Select **Print Setup** from the **File** menu to select the printer, paper size, source, and orientation. This opens the **Print Setup** dialog box.
2. Define the required parameters and click **OK.**

Runtime View Menu

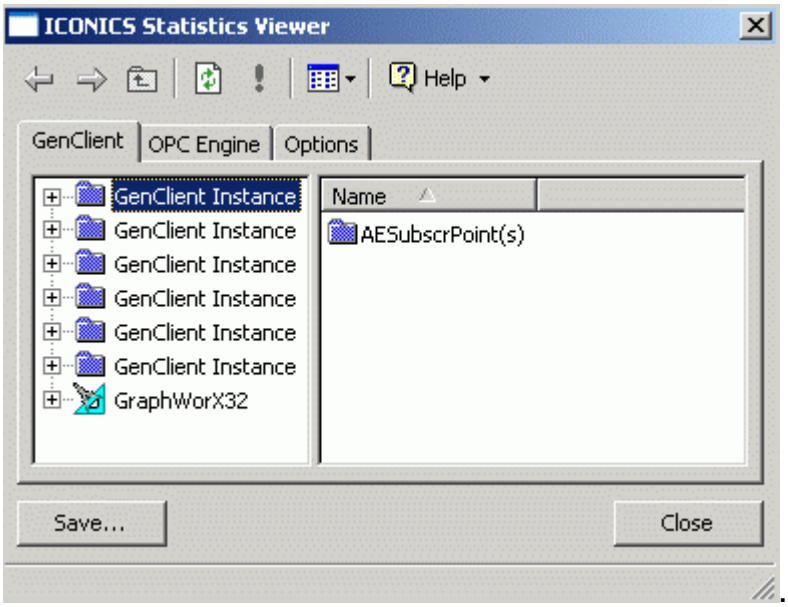
The **View** menu in runtime mode provides the same options as the **View** menu that is available in configuration mode, plus the following items.



Runtime View Menu

Data Statistics

Selecting **Data Statistics** from the **View** menu shows runtime information in the **Statistics Viewer** dialog, as shown in the figure below. This dialog is implemented in the GenClient/Olexpress DLL library and it is available throughout the GENESIS32 applications. For more information, please refer to the GenStatistics Viewer help documentation.



GenStatistics Runtime Viewer

Display History

GraphWorX32 maintains a history of the last 50 displays. You can browse through the display history using the commands **DisplayBack** and **DisplayForward** on the **View** menu. These commands are available as pick commands and as OLE Automation methods. The display history feature is similar to Microsoft Internet Explorer's back/forward feature.

Note: The display file history (forward/back commands) remembers the initial alias settings specified when a display is opened (see description of the runtime aliasing feature), and resets those aliases when you go back to that display in the file history.

Runtime Security

Access to the runtime environment of GraphWorX32 is restricted by the Security Server. Security levels are set initially in the project file and determine how much accessibility operators have during runtime. To login to runtime:

1. Select **Security Login** from the **Tools** menu. The **Security Login** dialog box opens.
2. Enter a user name and password in the appropriate fields, and then click **OK.**

Where there was insufficient security to view the current display, previous versions of GraphWorX32 shut down that display, and if there were no other windows or menus in the current display, the user was locked out by the Security Server without a way to log in as another user.

In the latest version of GraphWorX32, if a display is shut down by the Security Server, a separate display file called "Login.gdf" automatically opens, invoking the Security Login dialog box. In addition, GraphWorX32 displays the Security Login dialog in all cases where there is insufficient security.

Note: Please refer to the **ICONICS Security Configurator** documentation for more detailed information about security.

Runtime Data Entry

During runtime mode, you can quickly activate or enter data-entry fields by using the **TAB** key.

TraceWorX32 Support

Using a technology that has been incorporated into all ICONICS products, TraceWorX32 provides online diagnostics and tuning of applications running in the GENESIS32 system. TraceWorX32 is designed expressly for systems integrators, OEMs and customers who want to have tools for doing their own troubleshooting and diagnostics.

TraceWorX32 tracks the runtime activity for each GENESIS32 application and logs the runtime data to a log file based on user-configured trace levels. The log file provides a thorough, color-coded report detailing all activity for the application, including the time, the date, the severity level, and a description of the event or problem.

TraceWorX32 also features several options for reporting issues to technical support. If you are experiencing problems with any applications, the log file deployment options, such as compressing and e-mailing log files, are ideal for tracking and archiving data and sending detailed reports to technical support. Developers can use these reports to identify the source of the problems.

TraceWorX32 can be activated by the following registry keys for each GraphWorX32 component. The **ReleaseTraceLevel** can be set between 0 (full trace information) and 1000 (turned off). When it is active, a log file is generated either in the component's current directory, or in the subfolder of "Documents and Settings." Search for the *.log.xml document.

[HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\GWX32]

"ReleaseTraceLevel"=dword:00000000

[HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\GWXview32]

"ReleaseTraceLevel"=dword:00000000

For more information about TraceWorX32, please see the TraceWorX32 Help documentation.

OLE Express Compatibility

GraphWorX32 can imitate OLExpress 6.0 compatibility behavior. The patch fixes the jumping slider problem, but it also introduces the potential problem of data disintegrity. The feature is turned off by default.

Note that it can be optionally disabled for GWX32.exe by the following registry setting:

[HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\GWX32]

"CompatibilityOlexpress60Write"=dword:00000000

and for GWXview32.ocx by the following registry setting:

[HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\GWXview32]

"CompatibilityOlexpress60Write"=dword:00000000

The feature immediately updates other dynamics in the same display, so you can place a process point next to the slider and view the precise value you enter.

VCRWorX32 Data Replay

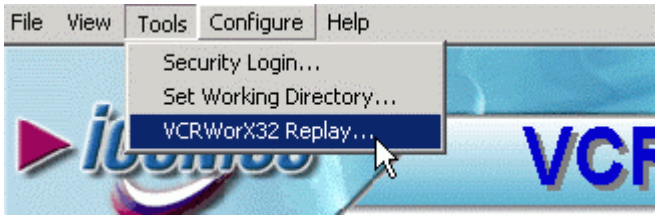
VCRWorX32 is a powerful tool in GENESIS32 that enables the replay and reanimation of graphical, historical and alarm data. Features of VCRWorX replay include:

The VCRWorX Control Panel, shown in the figure below, is the user interface that enables you to configure your historical data replay time and date settings. You can:

- See what the operator saw through replay of graphical displays.
- Specify time and date ranges for replay.
- Fast forward and rewind.
- Change the replay speed.
- Search for key events, tags, or expressions.
- Customize the layout and appearance using skins.

To launch the VCRWorX Control Panel from GraphWorX32 during runtime mode:

1. Select **VCRWorX32 Replay** from the GraphWorX32 **Tools** menu, as shown in the figure below.



Launching the VCRWorX Control Panel

2. This opens the **VCRWorX Control Panel,** as shown in the figure below. The GraphWorX display goes into VCRWorX historical replay mode based on the time and date you provided. You can see the process point values and ActiveX controls update in the GraphWorX display, and you can see the time/date field of the VCRWorX Control Panel in replay, or "playback," mode, as shown in the figure below.



VCRWorX Control Panel

For more information about VCRWorX, please see the VCRWorX help documentation.

Timeout Settings

GraphWorX allows customization of internal timeouts for several operations in the system registry:

[HKEY_LOCAL_MACHINE\SOFTWARE\ICONICS\GWX32\Runtime Settings]

"SecurityTimeoutSecs"=dword:0000003c

"LaunchAppTimeoutSecs"=dword:0000003c

"SystemWideMsgTimeout"=dword:0000003c

The default value of all timeouts listed above is 60 seconds.

SecurityTimeoutSecs - Specified timeout for obtaining security.

LaunchAppTimeoutSecs - Specified timeout for launching an external application.

SystemWideMsgTimeout - Specified timeout for sending system-wide message.

Chapter 11: VBA Wizards

Contents

Introduction to VBA Wizards
How the VBA Wizard Works
Rules for VBA Wizard
VBA Wizard Creation Tool
More Information About VBA Wizards
Other Sources of Information

Introduction to VBA Wizards

Visual Basic for Applications (VBA) Wizards are GraphWorX32 objects with Visual Basic code behind them. The code is run either in design mode to help to configure GraphWorX32 object or in runtime mode to execute a specific task.

Normally, the VBA code is stored in the current document, saved from the VBA Editor when the display is saved, and loaded back to VBA Editor when a display is open in GraphWorX32.

However, if the rules described below are followed, the VBA code can be "bound" to a GraphWorX32 object. When such an object is pasted / dropped to another instance of GraphWorx32, to the Symbol Library, or to scrap (desktop), this code goes with it.

How the VBA Wizard Works

The VBA Wizard can be run to perform a specific task either in design or runtime mode.

Design Mode

You can launch the VBA Wizard macro in design mode in GraphWorX32 by double-clicking the VBA Wizard.

By default, when an object is double-clicked in design mode, a Property Inspector is launched. However, if there is a special keyword in the first line of the **Custom Data** field of Property Inspector, a macro can be run.

The format of the keyword for a macro called "MacroName" is:

OnDoubleClick=<GwxMacroName_Main.MacroName>, Parameters=<>

There must exist a macro "MacroName" in the module "GwxMacroName_Main" in VBA to successfully run the macro. The user is allowed to put any string between angle brackets of a "Parameters=<>" section. Any data worth sending to the macro would be put between the angle brackets. These data are then available when the macro runs.

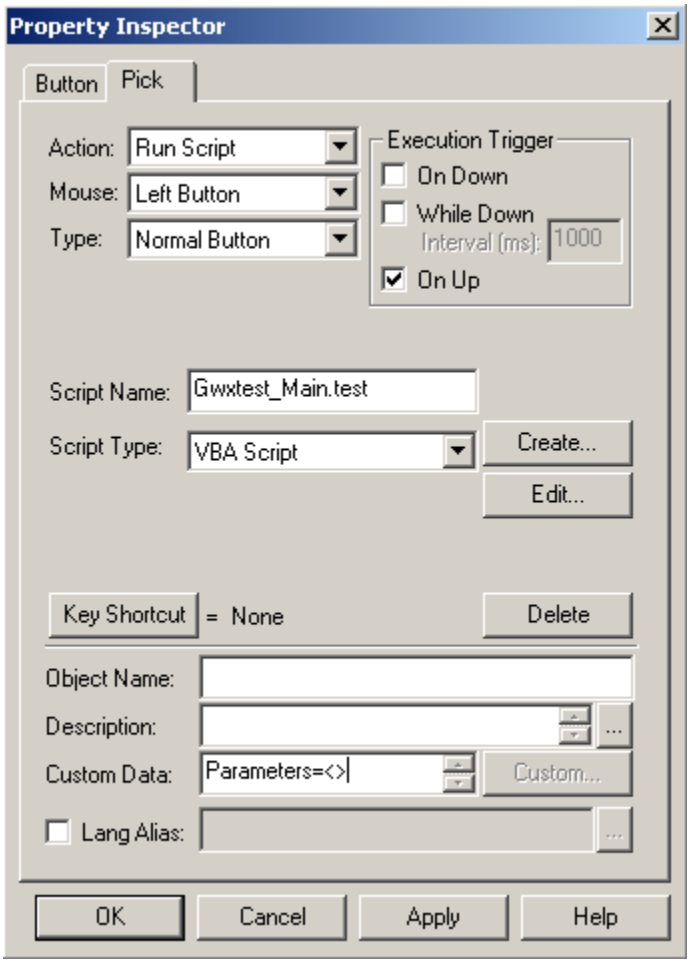
Note: The name of a macro cannot contain spaces

Runtime Mode

You can launch the VBA Wizard macro in runtime mode by clicking a GraphWorX32 button or a pick action configured to run a macro. The **Action** field in the **Pick** tab of the Property Inspector must be configured to **Run Script.** The **Script Name** field must contain a macro name in the following format:

GwxMacroName_Main.MacroName

Select **VBA Script** from the drop-down list under **Script Type.** Click the Create button to run the VBA Script Wizard.



VBA Wizard in Pick Action

The **Custom Data** field can contain any string keeping custom data as desired between angle brackets.

Rules for VBA Wizard

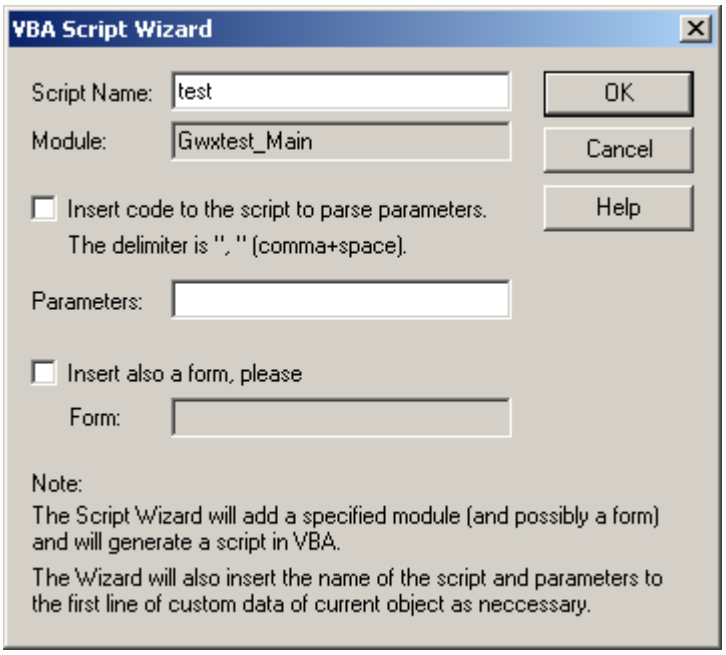
GraphWorX32 takes care of coding behind the VBA Wizard. Due to the special naming convention of all modules belonging to a specific VBA Wizard, the code in these modules can be properly moved with the symbol.

If the macro name of the VBA Wizard is "MacroName," then all code modules must start with the string "GwxMacroName_." This technique allows more code and form modules to be used for one VBA Wizard object, and it facilitates moving all of this code with the object when necessary.

VBA Wizard Creation Tool

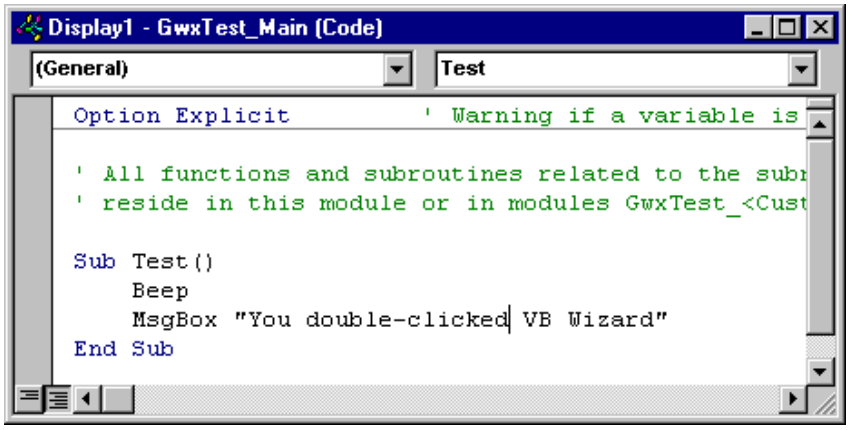
Because creating VBA Wizard objects can be a tedious task, GraphWorX32 offers a **VBA Script Wizard** that converts objects to a VBA Wizard and generates the VBA template code. The code can then be easily enhanced and modified. The following example demonstrates the use of this tool:

1. Open a GraphWorX32 display.
2. Create several ellipses and group them into a symbol.
3. Right-click on this symbol and select **Create VBA Script** from the popup menu. (You can also select **Macros > Create** from the **Tools** menu.) This opens the **VBA Script Wizard** dialog box, as shown below.
4. Enter a macro name (for example, "Test") in the **Script Name** field. Note, that the **Module** field is unavailable, but it is filled in automatically based on the name typed in the **Script Name** field.



VBA Script Wizard Dialog

5. Click **OK.** The Visual Basic Editor opens, as shown below, and the cursor should be placed in the body of the "Test" subroutine in a module "GwxTest_Main." You can now type the code, which is run when you double-click on the symbol in design mode. Try the example code shown in the figure below.



Test Subroutine (Generated by Wizard)

6. Return to GraphWorX32, and double-click the symbol. A beep should sound, and a message should appear.

You can try other examples with different settings in the following fields of the **VBA Script Wizard** dialog box:

Macro or Script Name
Module
Parameters
Insert Also a Form, Please
Form

Macro or Script Name

In the **Script Name** field of the **VBA Script Wizard** dialog box, type the name of your macro. The macro name should begin with a character and should contain alphanumeric characters only. If there is already a macro or a module of that name, you must choose another name.

Module

The **Module** field of the **VBA Script Wizard** dialog box is always grayed out because the module name is generated automatically based on the Script Name.

If the **Insert code to the script to parse parameters** check box is checked, some extra code is generated in the body of the macro subroutine. It helps in retrieving and storing parameters from the VBA Wizard object. This code uses the "GwxTools" module to convert your parameters to a string, called "StrPar," that is local to your macro's subroutine.

Parameters

Type any string you like in the **Parameters** field of the **VBA Script Wizard** dialog box. You can obtain this string when the macro runs. This optional field is designed to allow custom data specific to a VBA Wizard instance. Different instances of the same objects can keep different data. This feature serves to simplify your code by allowing you to pass values into a macro.

Insert Also a Form, Please

Check the **Insert also a form, please** check box in the **VBA Script Wizard** dialog box if you need a VBA form to be launched from the macro. You are allowed to create any number of forms for the VBA Wizard, assuming you follow the naming convention. (If you do not follow the convention, the VBA code is not moved with the object when necessary.)

Form

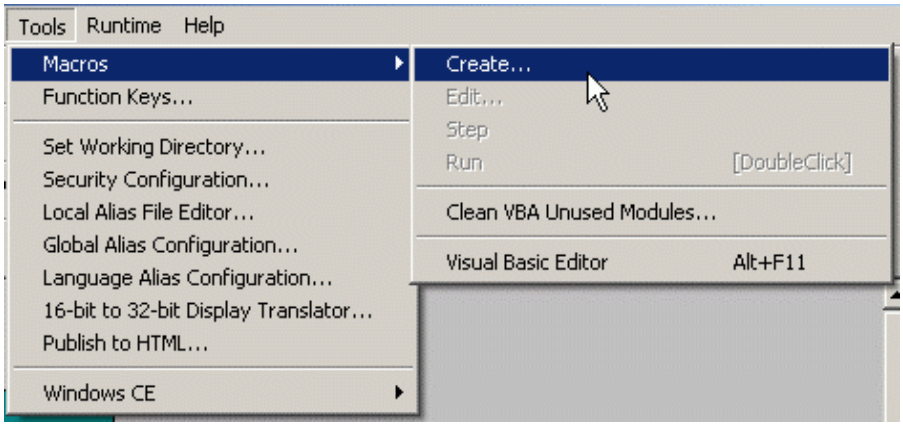
The **Form** field in the **VBA Script Wizard** dialog box is always grayed out and is generated automatically based on the Script Name.

More Information About VBA Wizards

Design Mode VBA Wizard
Runtime and Design Mode VBA Wizard
Runtime Mode VBA Wizard

Design Mode VBA Wizard

The **Macros** submenu of the **Tools** menu contains several other items commands for VBA Wizards in design mode, as shown in the figure below.



VBA Wizard Menus

Create Macro

Selecting **Macros > Create** from the **Tools** menu opens the **VBA Script Wizard** dialog box. You can also right-click on an object and then select **Create VBA Script** from the popup menu. Both menus are enabled if an object that is not VBA Wizard is selected in the display.

Edit Macro

Selecting **Macros > Edit** from the **Tools** menu opens the VBA Editor and places the cursor in the body of the macro. You can also right-click on an object and then select **Edit VBA Script** from the popup menu. Both menus are enabled if a VBA Wizard object is selected in the display.

Step Macro

Selecting **Macros > Step** from the **Tools** menu runs the macro in step (debug) mode. It opens the VBA Editor and stops on the first line of the macro. It is enabled if a VBA Wizard object is selected in the display.

Run Macro

Selecting **Macros > Run** from the **Tools** menu runs the macro. You can also double-click a VBA Wizard. It is enabled if a VBA Wizard object is selected in the display.

Runtime and Design Mode VBA Wizard

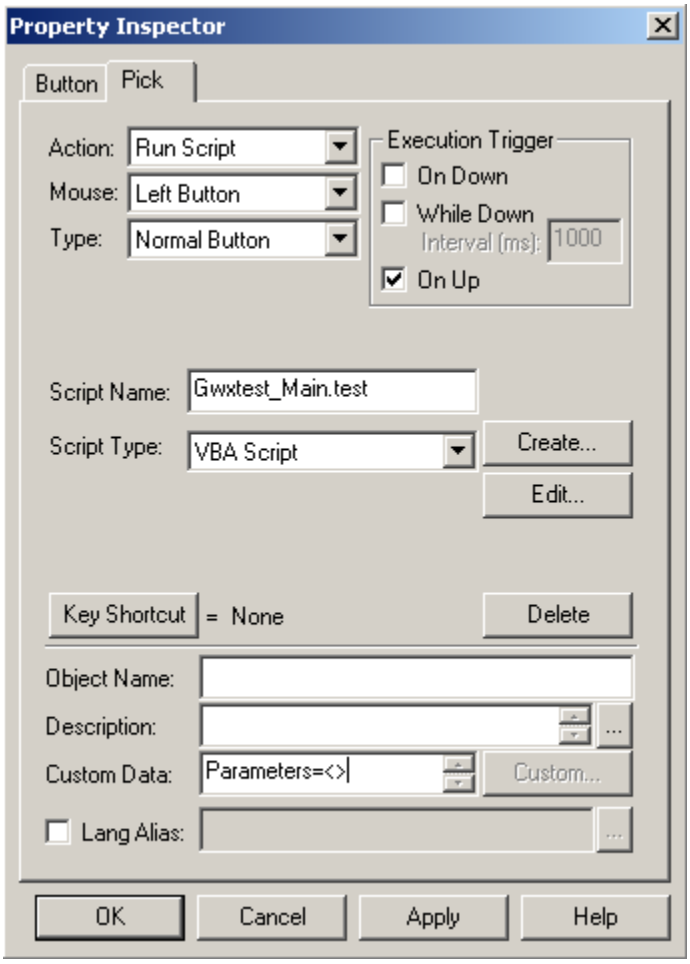
Clean VBA Unused Modules

When a VBA Wizard is deleted or moved out of the current display, the VBA code is not deleted automatically. However, selecting **Macros > Clean VBA Unused Modules** from the **Tools** menu removes all modules from the VBA Editor that start with the "Gwx" string and are not referenced from the currently displayed VBA Wizards (either design or runtime mode based).

Runtime Mode VBA Wizard

If you want to create a new runtime-based VBA Wizard, create a button or a pick action, select **Run Script** in the **Action** field, as shown in the figure below. Select **VBA Script** from the drop-down list under **Script Type.** Click the **Create** button to run the VBA Script Wizard.

Note: You do not need to use the **Edit** command to edit the macro. You can easily open the VBA Editor, find the module you want, and edit the macro directly.



VBA Wizard in Pick Action

Other Sources of Information

There are a lot of VBA Wizard objects packaged with GraphWorX32. You can find them either in sample displays or in the Symbol Library.

Chapter 12: GraphWorX32 ActiveX Controls

The GraphWorX32 Viewer ActiveX ("GWXview32.ocx") is an ActiveX control capable of running GraphWorX32 displays. The advantage of ActiveX controls like GraphWorX32 is that they can be embedded into any control container (e.g. Visual Basic Forms, Internet Explorer HTML pages, GraphWorX32 displays, etc.).

GraphWorX32 is essentially a runtime-only component (with some minimal configuration capability); the runtime-only design allows the ActiveX to be compact with respect to memory usage. The GraphWorX32 Viewer ActiveX executes displays created by GraphWorX32.exe. GraphWorX32 has all the runtime capabilities of GraphWorX32.exe, except the ability to execute VBA scripts.

Inserting the GraphWorX32 Viewer ActiveX

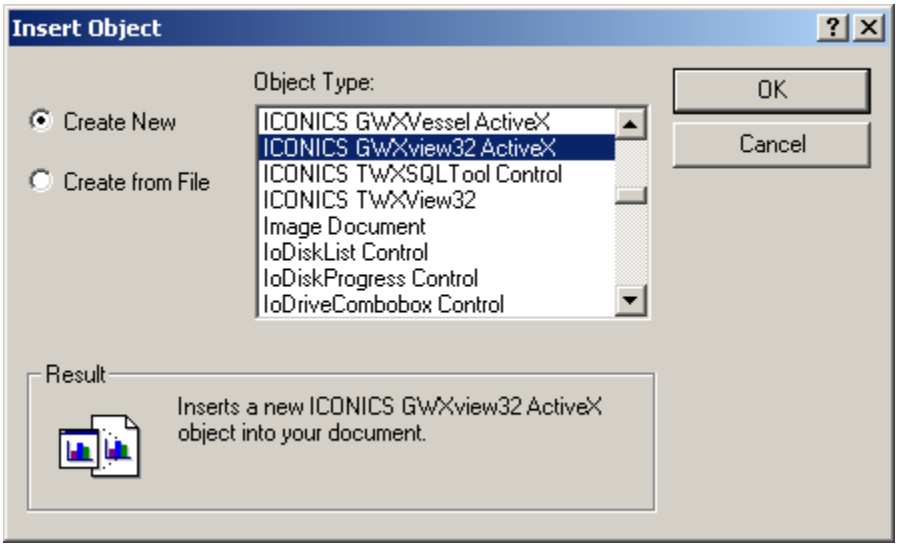
Techniques for inserting an ActiveX control may vary slightly among different control containers, however the basics are the same. This section describes how to insert the GraphWorX32 Viewer ActiveX into GraphWorX32.exe.

From the **Edit** menu, choose **Insert New Object,** or click the **Insert ActiveX Control/OLE Object** button on the **ActiveX** toolbar, as shown below.



Insert ActiveX Control/OLE Object Toolbar Button

This opens the **Insert Object** dialog box, shown in the figure below:



Insert Object Dialog Box

Select **ICONICS GWXView32 ActiveX** from the list of available controls, and then click **OK.** The GraphWorX32 Viewer ActiveX control will appear in the center of the display. Alternatively, you can click the **GWXView32** button on the **ActiveX** toolbar to directly insert the GraphWorX32 ActiveX.

ActiveX Toolbar

The GraphWorX32 ActiveX toolbar, which is part of the GraphWorX32 Screen, is shown below.



GraphWorX32 ActiveX Toolbar

Customizing the ActiveX Toolbar

Any ActiveX that is inserted in the display via the **OLE** button on the ActiveX toolbar is automatically added to the toolbar as a new button. You can remove ActiveX buttons from the toolbar by right-clicking on the toolbar. This opens the **Customize Toolbar** dialog box. Select the ActiveX from the list on the right, and then click **Remove.** Alternatively, you can remove a button from the toolbar by clicking the button on the toolbar while pressing the shift key and dragging the button off the toolbar.

If you want a certain set of ActiveXs to be available after installation, you can use the following registry settings to define a starting set of ActiveXs for the toolbar.

HKEY_LOCAL_MACHINE\Software\Iconics\Gwx32\OEM Information\ActiveX Toolbar Default Info\Number of ProgIDs

This is the number of ActiveX buttons that will appear on the toolbar.

HKEY_LOCAL_MACHINE\Software\Iconics\Gwx32\OEM Information\ActiveX Toolbar Default Info\ProgID_1

HKEY_LOCAL_MACHINE\Software\Iconics\Gwx32\OEM Information\ActiveX Toolbar Default Info\ProgID_2

HKEY_LOCAL_MACHINE\Software\Iconics\Gwx32\OEM Information\ActiveX Toolbar Default Info\ProgID_3

...etc.

These are Programmatic IDs of the ActiveXs to appear on the toolbar (for example, "GWXVIEW32.GWXview32Ctrl.1").

Note: These registry entries will not do anything if GraphWorX32 has already been run (because the currently saved toolbar settings override these initial settings). If GraphWorX32 has already been run, you would need to delete the following registry entry:

HKEY_CURRENT_USER\Software\Iconics\Gwx32\Custom Toolbar Settings.

OLE ActiveX

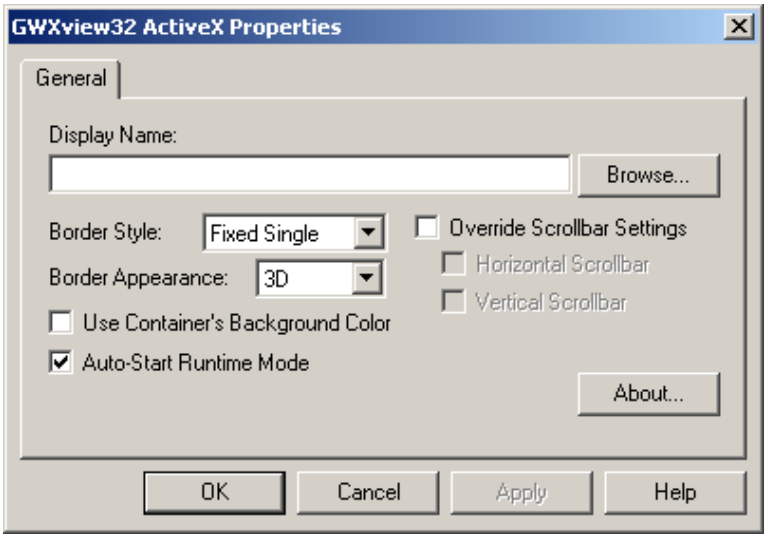


Clicking the **OLE** button on the **ActiveX** toolbar inserts an ActiveX Control/OLE Object.

Graphics ActiveX



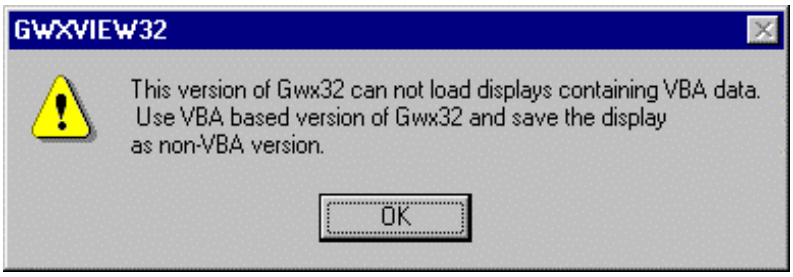
Clicking the **GWXView32 ActiveX** button on the **ActiveX** toolbar opens the Graphics ActiveX.



GWXView32 ActiveX Properties Dialog Box

The dialog box shown above allows you to insert a GraphworX32 Display within this display.

You can save this inserted display with or without VBA. The default is that is saved with VBA. However, if the display is embedded, is should be saved without VBA as the following error message will display if you attempt to save it with VBA.



Error Message

Trend ActiveX



Clicking the **TWXView32** button on the **ActiveX** toolbar opens the TrendWorX32 Viewer ActiveX. For more information, refer to the TrendWorX32 Viewer ActiveX help documentation.

Alarm ActiveX

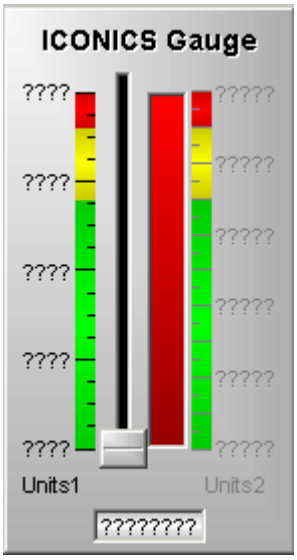


Clicking the **AWXView32** button on the **ActiveX** toolbar opens the AlarmWorX32 Container. For more information, refer to the AlarmWorX32 Viewer ActiveX help documentation.

Gauge ActiveX



Clicking the **GWXGauge ActiveX** button on the **ActiveX** toolbar opens the Gauge ActiveX, shown below.



ICONICS Gauge

For more information about the Gauge ActiveX, refer to the **GWXGauge ActiveX** help documentation.

Switch ActiveX



Clicking the **GWXSwitch ActiveX** button on the **ActiveX** toolbar opens the Switch ActiveX, shown below.



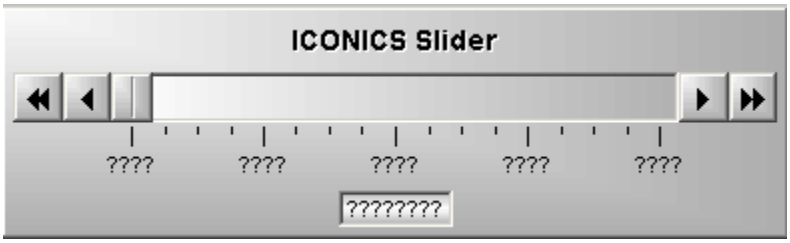
ICONICS Switch

For more information about the Switch ActiveX, refer to the **GWXSwitch ActiveX** help documentation.

Slider ActiveX



Clicking the **GWXSlider ActiveX** button on the **ActiveX** toolbar opens the Slider ActiveX, shown below.



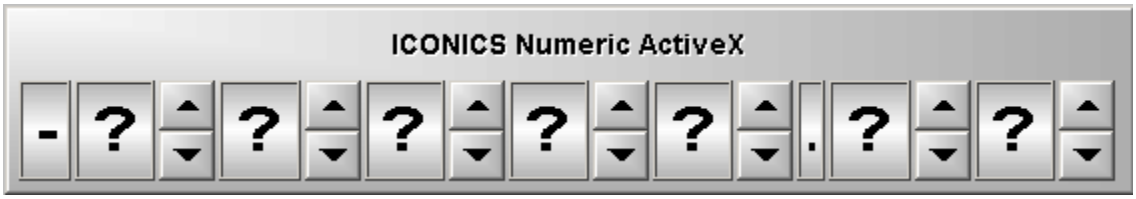
Slider ActiveX

For more information about the Slider ActiveX, refer to the **GWXSlider ActiveX** help documentation.

Numeric ActiveX



Clicking the **GWXNumeric ActiveX** button on the **ActiveX** toolbar opens the Numeric ActiveX, shown below.



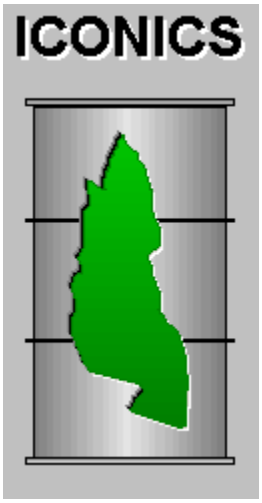
Numeric ActiveX

For more information abut the Numeric ActiveX, refer to the **GWXNumeric ActiveX** help documentation.

Vessel ActiveX



Clicking the **GWXVessel ActiveX** button on the **ActiveX** toolbar opens the Vessel ActiveX, shown below.



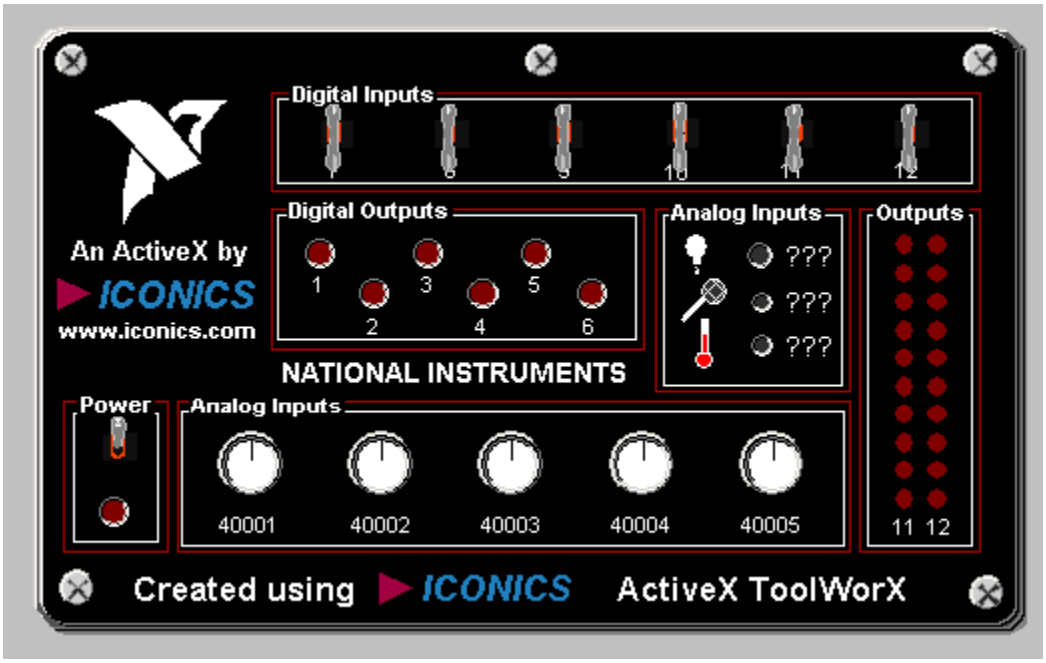
Vessel ActiveX

For more information about the Vessel ActiveX, refer to the **GWXVessel ActiveX** help documentation.

National Instruments ActiveX



Clicking the **National Instruments ActiveX Control** button on the **ActiveX** toolbar opens the National Instruments ActiveX Control, shown below.



National Instruments ActiveX Control

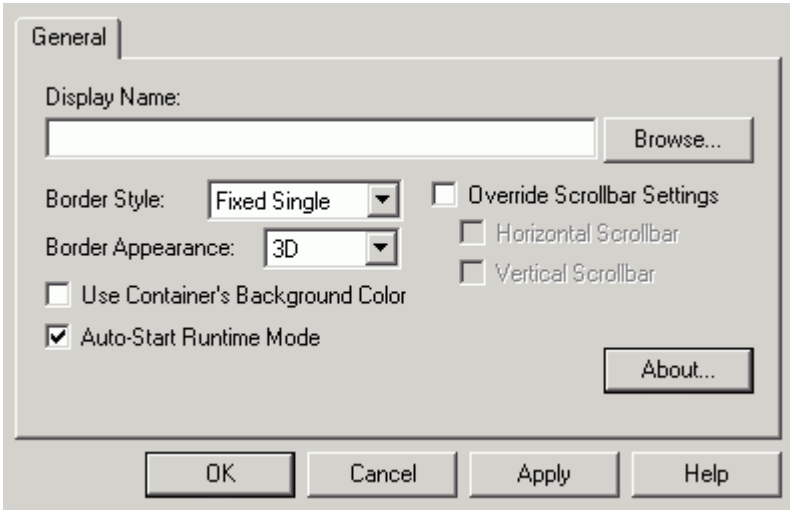
For more information about the National Instruments ActiveX, refer to the **National Instruments ActiveX** help documentation.

Configuring the GraphWorX32 Viewer ActiveX

This property page allows you to change certain attributes of the GraphWorX32 Viewer ActiveX control.

To configure the GraphWorX32 Viewer ActiveX:

1. Double-click the GraphWorX32 Viewer ActiveX to display the control's **Properties** dialog box, shown below:
2. Fill in the parameters described in the **GraphWorX32 Viewer ActiveX Properties** section below, and then click **OK.**



GraphWorX32 Viewer ActiveX Properties Dialog Box

GraphWorX32 ActiveX Parameters

Parameter	Description
Display Name	Specifies the file name of the GraphWorX32 display to be loaded in this control. Click the **Browse** button to search for display files.
Border Style	Specifies the appearance of the window border: "No Border" or "Fixed Single."
Border Appearance	Specifies the appearance of the window border: flat or three-dimensional.
Use Container's Background Color	When this option is checked, the background color of the control is automatically set to match the background color of the container in which this control is embedded. This property only works if the container supports the "AmbientBackColor" property. GraphWorX32 and Visual Basic Forms both support this property.
Auto-Start Runtime Mode	When checked, the control will automatically enter runtime mode when the control's container enters runtime mode. This property only works if the container supports the AmbientUserMode property. Both GraphWorX32 and Visual Basic Forms

	support this property.
Override Scrollbar Settings	When this option is checked, this property indicates the control will override the scrollbar visibility settings of the display currently loaded in the control. When it is not checked, the control will use the scrollbar settings defined in the currently loaded display.
Horizontal Scrollbar	When this option is checked, the horizontal scrollbar of the control's window is visible. When it is not checked, the horizontal scrollbar is hidden. This property is ignored if **Override Scrollbar Settings** is not checked.
Vertical Scrollbar	When this option is checked, the vertical scrollbar of the control's window is visible. When it is not checked, the vertical scrollbar is hidden. This property is ignored if **Override Scrollbar Settings** is not checked.

Chapter 13: OLE Automation Reference

Contents

Introduction
Registration at GenRegistrar in Design Mode
Object Hierarchy
Automation Programming in C++ and Visual Basic: Understanding Inheritance
GraphWorX32 OLE Automation Example Displays
Events
Properties and Methods

Introduction

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.

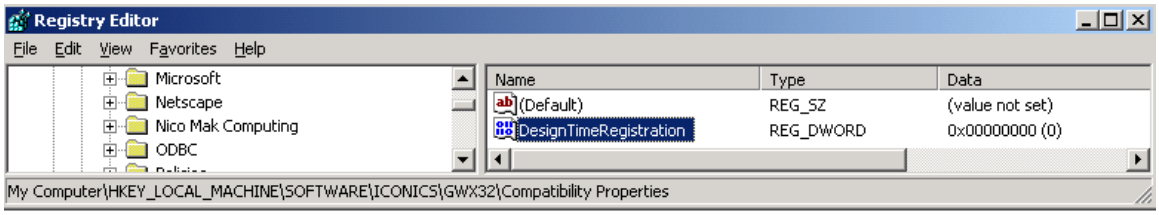
Registration at GenRegistrar in Design Mode

GraphWorX32 also registers at GenRegistrar in design mode. In design mode, GraphWorX32 registers the dispatch pointer to the display with a keyword “GWX32_DESIGN_MODE”.

When entering runtime mode, it re-registers the dispatch pointer to <current-display-title> and keeps this key as long as it is in runtime, regardless of what other displays are loaded. This provides an easy way to recognize miscellaneous instances of GraphWorX32 in runtime (based on the caption with which they started).

The Design Time Registration feature is by default off, and it can be turned on using the registry key DesignTimeRegistration under GWX32 / Compatibility Properties, as shown in the figure below:

0 feature disabled
1 feature enabled



Enabling Design Mode Registration at GenRegistrar

When leaving runtime mode, GraphWorX32 registers again with the key “GWX32_DESIGN_MODE”.

The following example shows how an instance of GraphWorX32 in design mode can be obtained from a Visual Basic application:

```
Sub AutomateGwx()
    ' Note: Add ICONICS AutoGenRegistrar to Project
    '   | References
    ' Note: Add GWX32 to Project | References

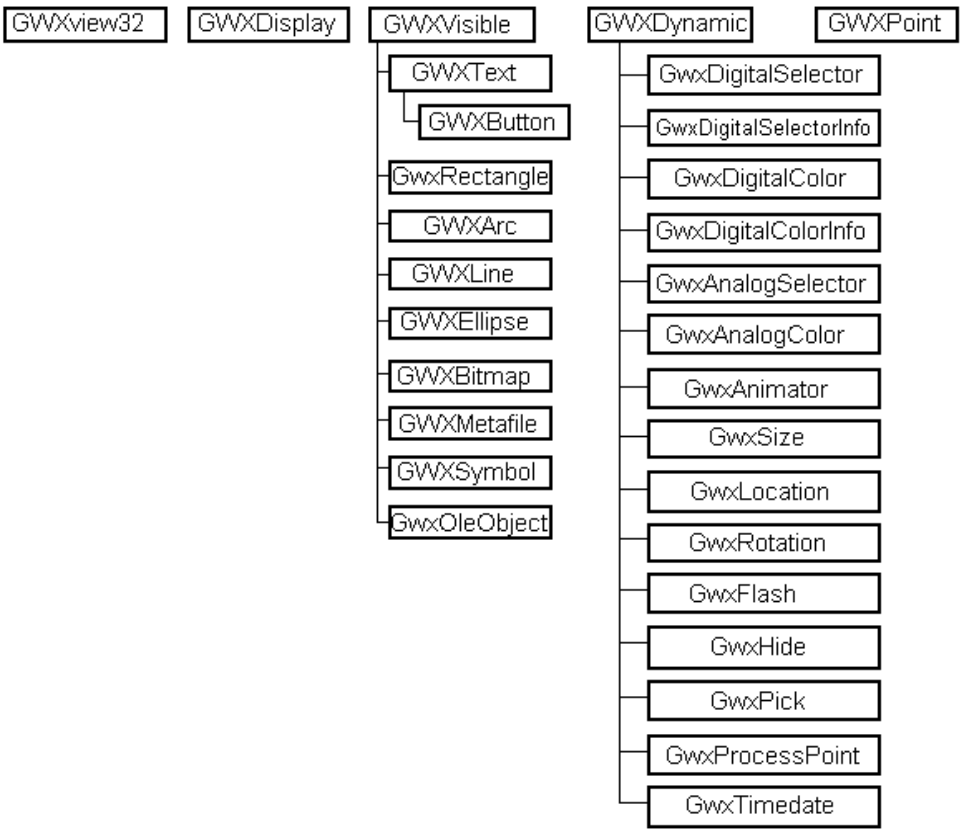
    ' Create an Automation GenRegistrar Wrapper
    Dim genreg As New AutoGenRegistrar
    If genreg Is Not Nothing Then Exit Sub
    ' failure

    'Declare GWX reference and get it from
    'GenRegistrar
    'Note that we ask for GWX instance in design mode
    Dim gwx As Gwx32.GwxDisplay
    Call genreg.GetDispatch("GWX32_DESIGN_MODE", "GWX32", "*",
gwx)
    If gwx Is Nothing Then Exit Sub     ' failure

    ' Automate GWX instance
    gwx.ShowWindow
    gwx.BringWindowToTop
    Randomize
    gwx.BackgroundColor = RGB(Rnd * 255, Rnd * 255, Rnd * 255)
```

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:

```
            "\\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


Status = ThisDisplay.ReplaceHostEx("Host1", "Host2", True, True)
'if Status <> 0 then there was no replacement performed or an error occurs


 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, _
                    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, _

                        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.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


    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, _

                            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

                        "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."


    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

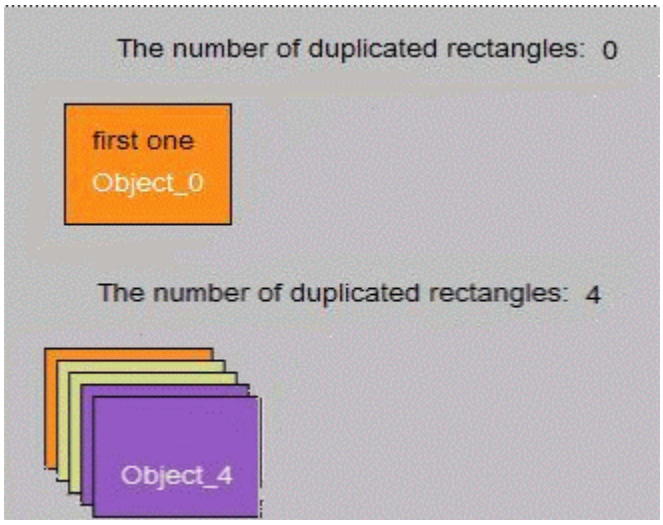
```
Sub DuplicateSelection()
```

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.

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


Select Case lFileVersion
    Case -1
```

```
            strVersion = "OpenFileError"
        Case 0
            strVersion = "Ver520"

            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


    '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, _

                          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:

```
'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, _

'#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 " + _

        Chr(13) + "Click on the button once more please."
```

VB example:

```
Call ThisDisplay.OpenPopupWindow("PopupWindow.gdf", _
                                 False, _
                                 True, _

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

    popup.GetFileName
End If
```

VB example:

```
'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

list = obj.objectName

'Keep going ...
Do
    counter = counter + 1

        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

    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!")
```

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), _

            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

    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


    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


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


    '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"

        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
                                    "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: " + _

    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

    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, _

                                                0.5)
    MsgBox "The window pans the view on submit."

    Call ThisDisplay.ViewPan(70, 200)



    'Restore window dimensions.

                                                1 / 0.0025, _
                                                1 / 0.45, _

    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."



    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)
```

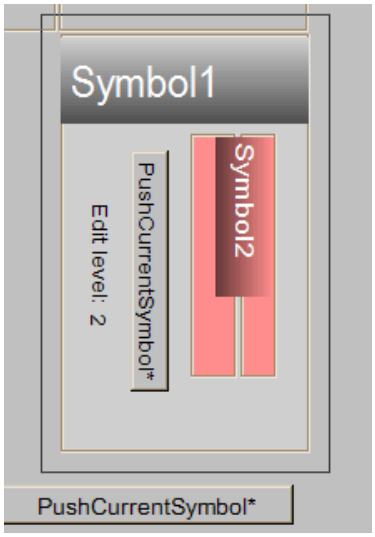
Function PushCurrentSymbol

```
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"
```

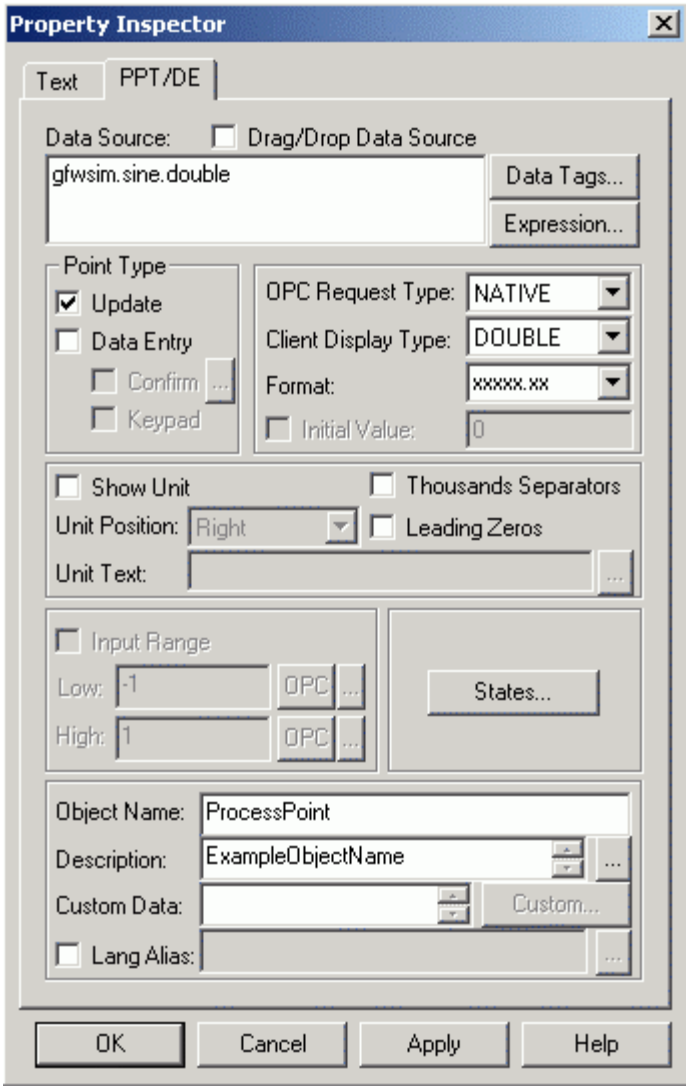
VB example:

```
Dim obj
Set obj = ThisDisplay.GetDynamicObjectFromName("ProcessPoint")

Dim strName
strName = obj.GetObjectTypeName

obj.UserDescription = "ExampleDataSource"


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


 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)

        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 = _


 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.

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.

Chapter 14: 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.

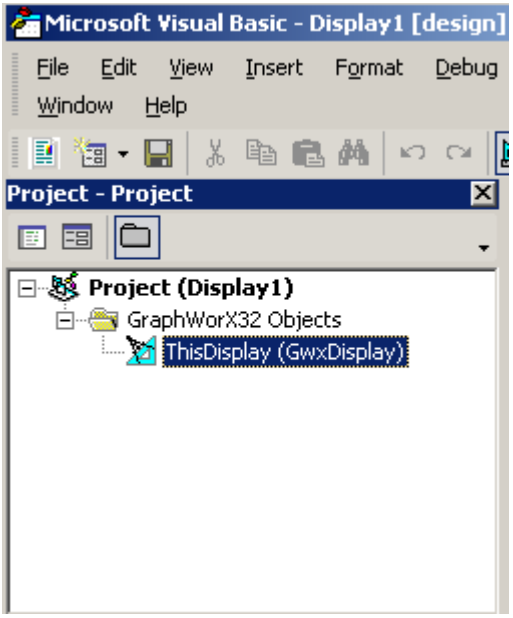
For more information about VBA, please see the GENESIS32 VBA Tutorial or the Microsoft VBA Help documentation.

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

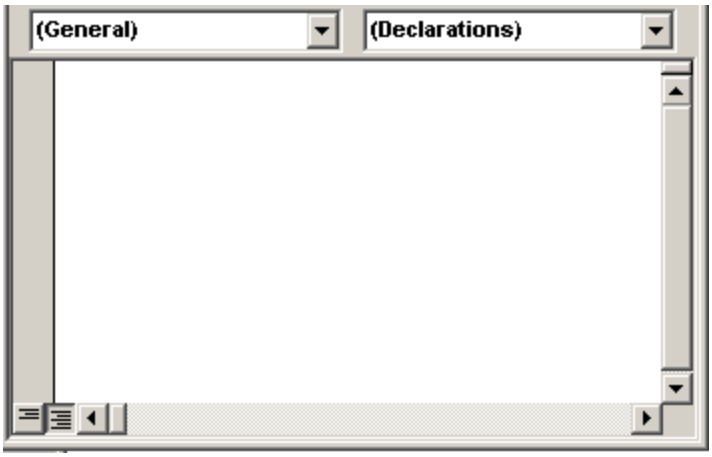
ThisDisplay is a special module representing the current GraphWorX32 display.



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

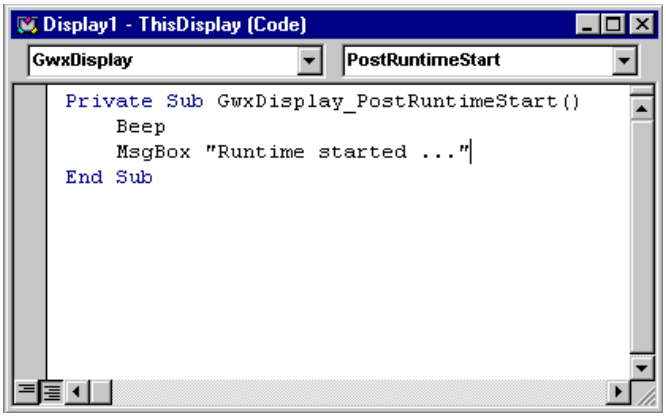
Event handlers can be added for a display using VBA. To do this, you should have the display loaded on the screen.

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

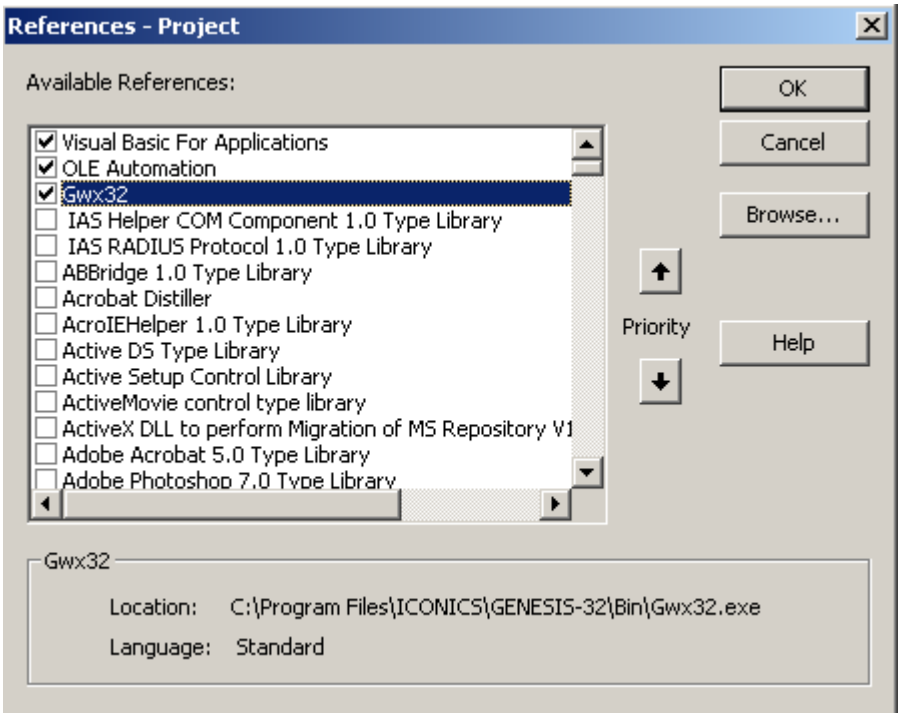
4. Switch to GraphWorX32 either by closing the VBA Editor or by pressing the **ALT+TAB** keys.
5. Test the event by starting runtime by clicking **Runtime** on the GraphWorX32 menu bar or by pressing the **CTRL+M** keys.
6. You should hear a beep, and a message box with a message "**Runtime started**" 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.

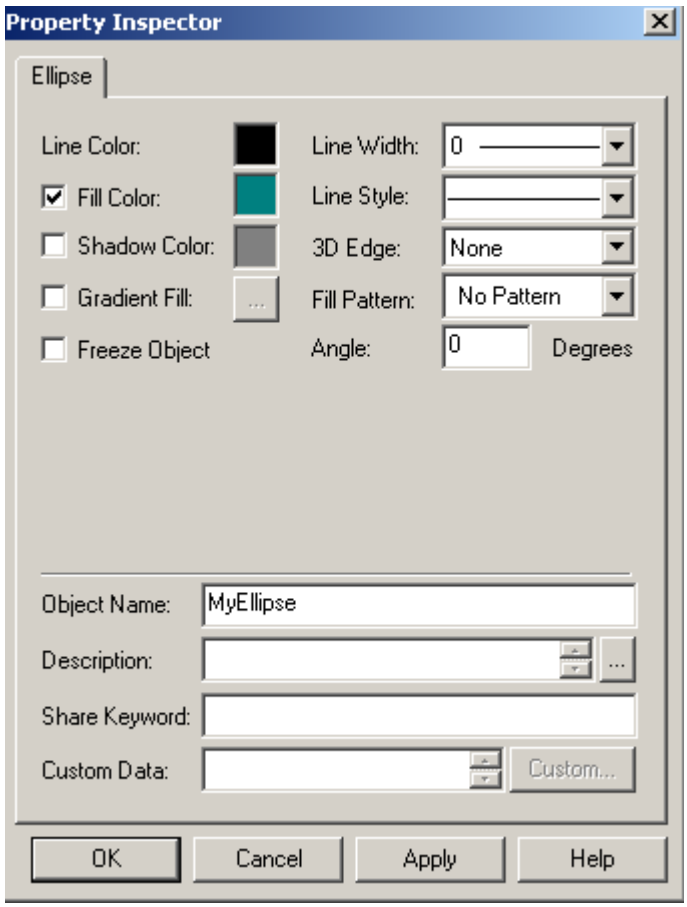
```
Dim o_Vis As GwxRectangle
Dim o_Dyn As Object
Set o_Vis = ThisDisplay.GetVisibleObjectFromName("rect")
Set o_Dyn = o_Vis.GetDynamicObjectFromIndex(1)
o_dyn.lowRange = 10
```

***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

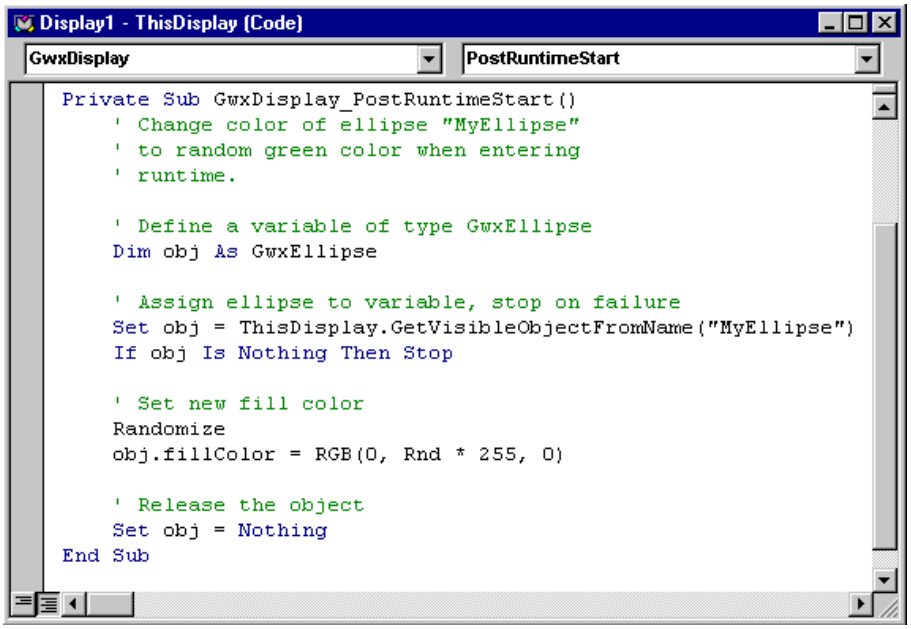
Write code for changing the color of an ellipse GraphWorX32 native object during Runtime.

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.**

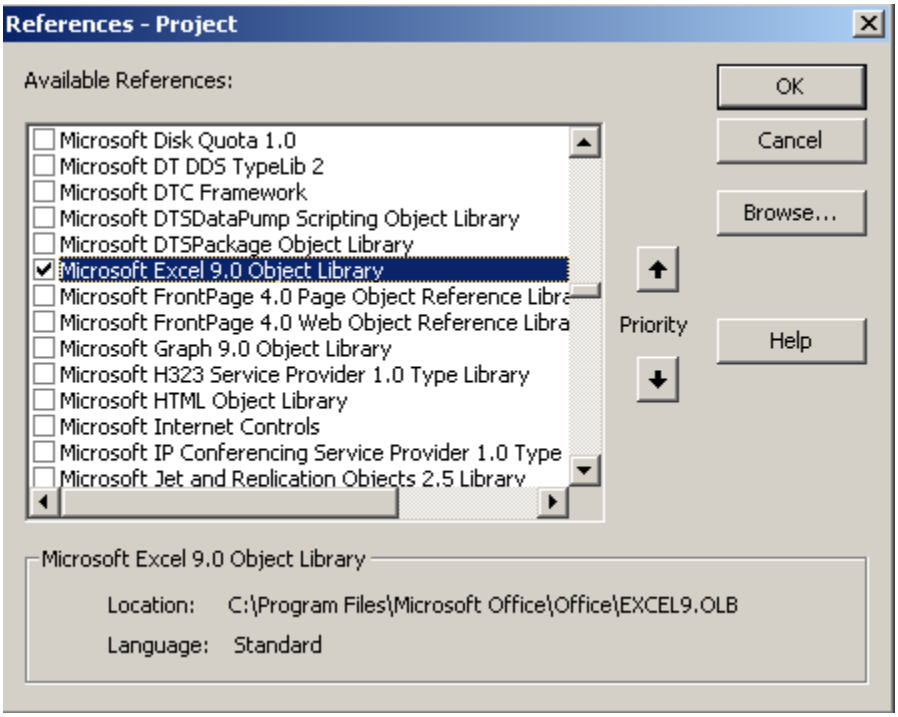
Using VBA to Connect With Other Applications

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.

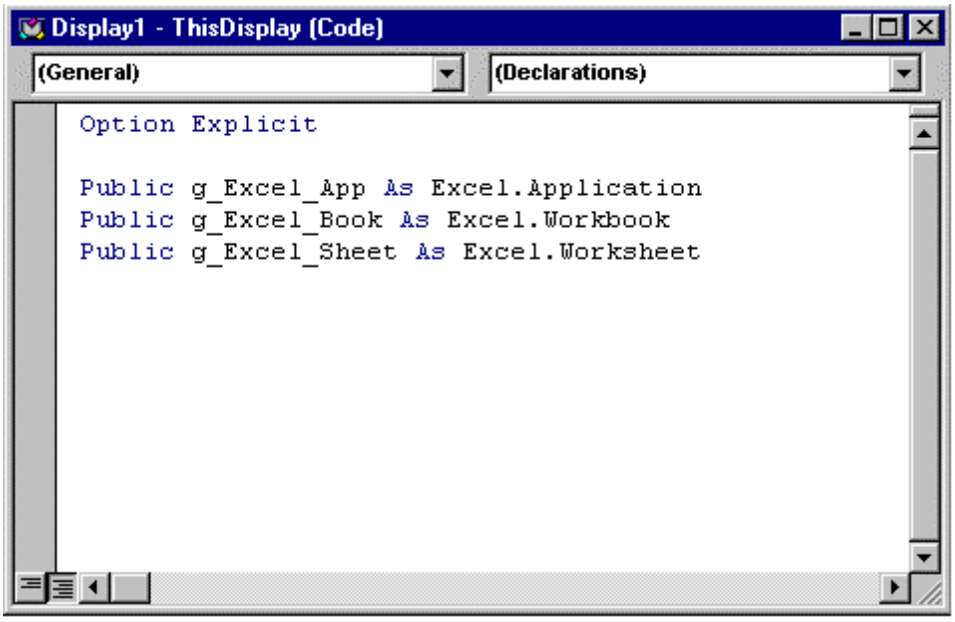


Selecting Microsoft Excel in the References Dialog

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
```

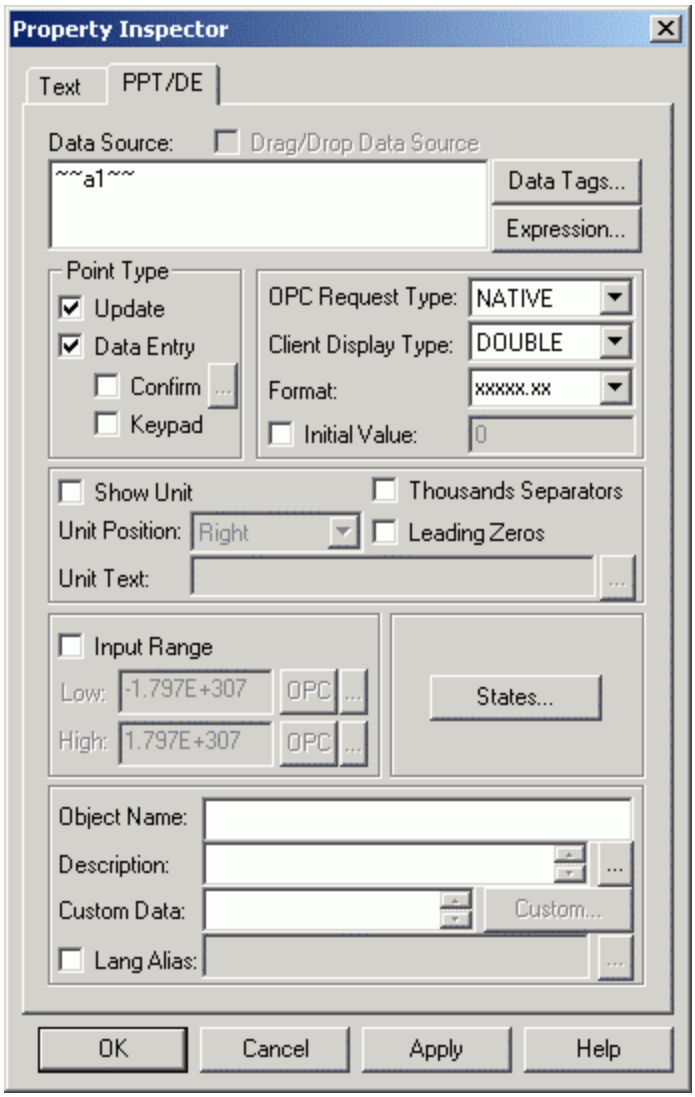


ThisDisplay Code Window With Global Declarations

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.