How to Use API Objects in VBA Programs

Topic Under Construction!!!

Before you can use any iGrafx Professional API object in your application, you must add the OBJMODL1.TLB file to your project. If you use the object in most of your Visual Basic projects, you may want to add it to Visual Basic's Autoload file.

To distribute applications you create that use any of the iGrafx Professional API objects, you must install and register it on the user's computer. The Setup Wizard provided with Visual Basic provides tools to help you do that. Please refer to the Visual Basic manual for details.

What are Objects and Controls

An object is simply a combination of code and data that can be treated as a single unit. An object can be an entire application or any individual piece of an application. A control is an object; however, a control has a specific meaning. Essentially, a control is an object that is contained within a form object, and the combination of forms and control make up the visible interface to the application. For instance, a command button (OK, Cancel) is a control; the application object is not a control for two reasons: it does not exist within a form (rather, the form object exists within the application object), and it is not a visible interface component.

An object is defined by a class. Think of a class as a template that defines the general characteristics of an object. In this sense, an object can be almost anything, from an interface form or button to a stylized computer representation of a ball or balloon. A class is generic; an object is a specific instance of a class. For example, a dialog box may have two command buttons on it: one to OK the event, and one to Cancel the event. Both buttons are defined by the CommandButton class and have all the same properties, methods, and events. Each individual button is an object for which the values of its properties, the actions taken by its methods, and the responses to its events can all be different.

Using the iGrafx Professional API Reference

This section of the iGrafx Professional Help documentation provides all the information you need to develop VBA code that works with the features of iGrafx Professional. The API consists of objects and controls , which in turn consist of properties, methods, and events. In addition, the API provides a set of named constants (also called enumerated types) that are used to provide values for specific properties.

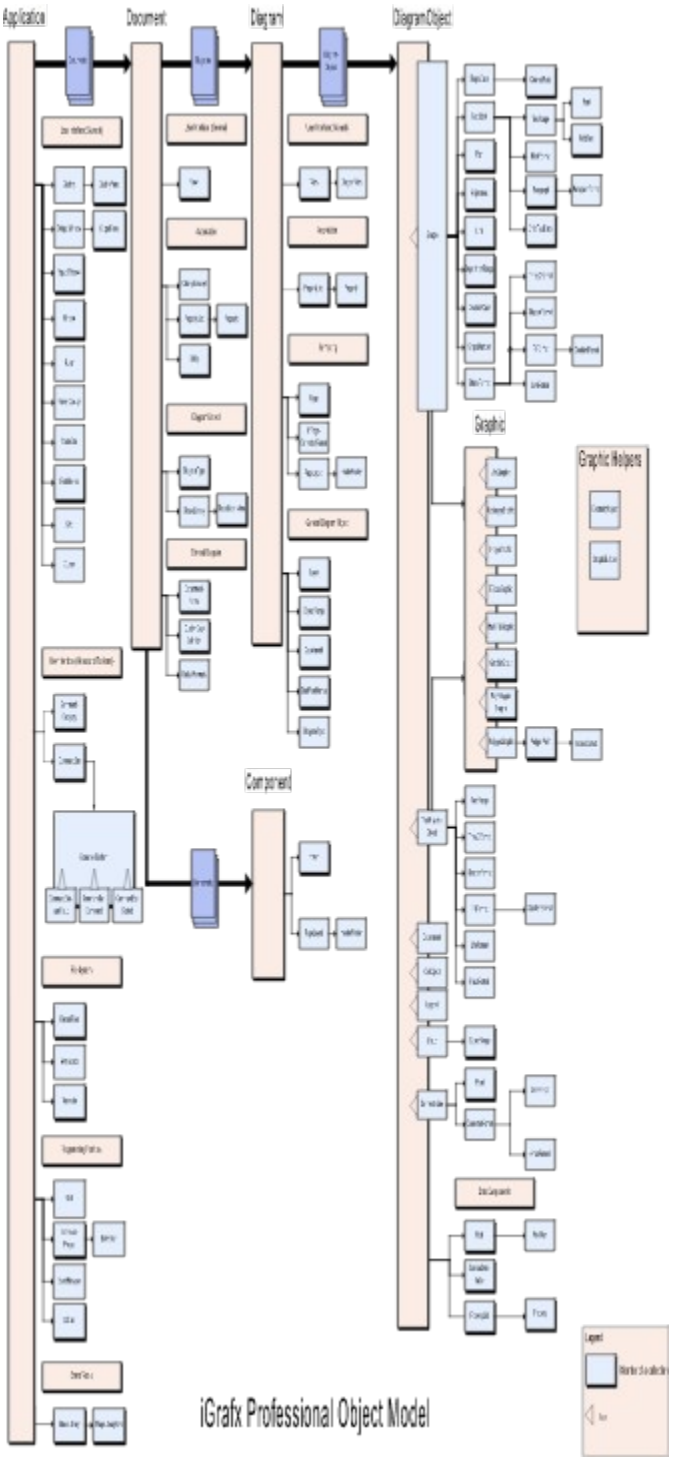
What are Properties, Methods, and Events

Properties, methods, and events define the characteristics of an object. Properties can be thought of as the attributes of an object. For instance, if a ball is the object, some attributes of the ball would be its color, its size, the type of material it is made from, and so on. A method can be thought of as the actions that the object can perform. For instance, the ball object can inflate and deflate. An event can be thought of as the external events that can affect an object. For instance, for the ball object an event could be a puncture or a kick.

iGrafx API Object Hierarchy

The object hierarchy for iGrafx Professional and iGrafx Process is shown in the following illustration. The illustration focuses solely on the hierarchy of objects, and their relationship to one another. The illustration is not meant to represent every aspect of each object; rather, it is only meant to show the chain of subordination from one object to another.

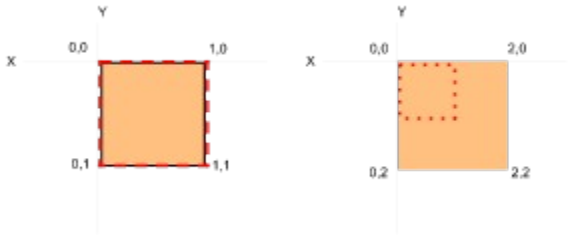
See the Legend for information about the meaning of colors and other indicators used in the illustration.



Shape Coordinate Space

The coordinate space of a shape is typically the bounding rectangle of the shape. The coordinate space is always relative to the shape, and ranges in value from 0 to 1 in both the X and Y directions. All actions, events, etc., for a shape occur in its relative coordinate space. For instance, if the X and Y positions of a mouse click are the input parameters to an event procedure, the parameter values are based on the relative coordinate space of the shape.

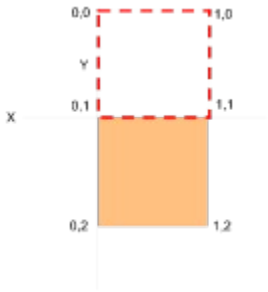
The following diagram shows the normal “default” arrangement for a simple shape and its coordinate space on the left. On the right is shown the effect of reducing the size of the shape’s coordinate space.



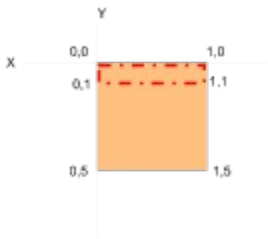
For the shape on the left, the top left corner is at 0,0; the bottom right corner is at 1,1; the center point of the shape is at .5, .5. For the shape on the right, the top left corner is at 0,0, the bottom right corner at 2,2, and the center point at 1,1.

There are various reasons for using a custom coordinate space, but the most common is to make some aspect of programming with adjustments easier. Using the custom coordinate space shown in the previous illustration allows the programmer to write code that enforces, for instance, that an adjustment point stays between 0,0 and 0,1 instead of 0,0 and 0,.5. Although this doesn’t provide any marked advantage for this simple case, in more advanced iShapes, custom coordinate spaces provide other advantages.

The position of the shape’s coordinate space also can be adjusted; however, the benefits of doing this are questionable. For instance, consider the effect of moving the coordinate space of a shape so it is above the shape in the Y direction, as shown below. Notice that this simply doubles the numeric values in the Y direction.



One way to think about the shape coordinate space is that it defines a ‘unit’ area. Lets look at one more example. In the following illustration, the relative coordinate space has been reduced in the Y direction to one-fifth the size of the shape. The unit area is still one unit for X and one for Y, but in this case 5 unit areas are needed to fill the shape in the Y direction.



Important Notes:

- A shape’s coordinate space (size and location) is set through the Graphic object’s SetCoordinateSpace method (Shape.Graphic.SetCoordinateSpace). The coordinate space can be reset to the default (the shape’s perimeter) with the Graphic object’s ResetCoordinateSpace method (Shape.Graphic.ResetCoordinateSpace).
- The coordinate space has nothing to do with mouse click “hit” events. That is, if the coordinate space is placed completely outside of the shape, the user still must click within the shape’s boundaries for the click to be recognized as belonging to the shape (and not the diagram).
- Main use of the coordinate space is for the Adjustment and Graphics objects.

Object Properties

Some objects within the iGrafx API have properties that return another object as its data type. These are called Object Properties, and they provide access to the various levels of the API Object Hierarchy. In essence, object properties provide a traversal mechanism that allows the developer to “get to” all of the various object types.

Object properties are read-only, and have the following characteristics:

- They are used to access the properties of an existing instance of the object type. The accessed object may have “data-value properties” which can be read or assigned values (properties with data types such as Variant, String, Integer, etc.), or it may have additional object properties that, in turn, lead to other levels of the object hierarchy.
- They are used to invoke any methods belonging to the object type.

The key point is that object properties cannot be used as assignment targets. For example, if Diagram1 is the active diagram, you cannot reset what the active diagram is with an assignment statement as follows:

```
Application.ActiveDiagram = Diagram3
```

To actually make Diagram3 the active diagram rather than Diagram1, you would use the Diagram object’s ActivateDiagram method as follows:

```
Application.Document1.Diagram3.ActivateDiagram
```

API Event Model

Topic Under Construction!!!

Activation/Deactivation Events

Event Name	Used By Object	Description
Activate	Application	
	Document	
	Component	
	Diagram	
Deactivate	Application	
	Document	
	Component	
	Diagram	

Start and Stop Events

Event Name	Used By Object	Description
Quit	Application	
Initialize	DiagramType	
	Extension	
	ShapeClass	
Terminate	DiagramType	
	Extension	
	ShapeClass	
Startup	Application	

File-Related Events

Event Name	Used By Object	Description
NewDocument	Application	
OpenDocument	Application	
New	Document	
	Component	
	Diagram	
	DiagramObject	
Open	Document	

	Component
	Diagram
Save	Document
	Component
	Diagram
Print	Component
Close	Document
	Component
	Diagram
	DiagramObject

Modification Events

Event Name	Used By Object	Description
Move	Application	
Resize	Application	
Modify	Document	
	DiagramObject	
Delete	Component	
	Diagram	
DeleteObject	DiagramObject	
Rename	Component	
	Diagram	
	Department	
AdjustmentMove	Shape	

Selection Events

Event Name	Used By Object	Description
Select	DiagramObject	
Deselect	DiagramObject	
SelectionChange	Diagram	

Object “Change” Events

Event Name	Used By Object	Description
DepartmentChangeName	Document	
PropertyChange	Document	
	Diagram	
	DiagramObject	

PageLayoutChange	Diagram
CustomDataDefinitionChange	DiagramType
ChangeDepartment	Shape

“Before” an Action Events

Event Name	Used By Object	Description
BeforeWelcome	Application	
BeforeKeyDown	Application	
	Diagram	
BeforeClose	Document	
	Diagram	
BeforePrint	Document	
	Diagram	
BeforeSave	Document	
	DiagramObject	
BeforeClick	Diagram	
	DiagramObject	
BeforeDoubleClick	Diagram	
	DiagramObject	
BeforeRightClick	Diagram	
	DiagramObject	
BeforeChangeLayer	DiagramObject	
BeforeDelete	DiagramObject	
BeforeEditCustomData	DiagramObject	
BeforeGroup	DiagramObject	
BeforeMove	DiagramObject	
BeforeRotate	DiagramObject	
BeforeSelect	DiagramObject	
BeforeSize	DiagramObject	
BeforeUngroup	DiagramObject	
BeforeAdjustmentMove	Shape	
BeforeConnectorAttach	Shape	
BeforeConnectorDetach	Shape	
BeforeExecuteLink	Shape	
BeforeFontChange	Shape	
BeforeReplace	Shape	
BeforeStyleChange	Shape	
BeforeTextChange	Shape	
BeforeAttach	ConnectorLine	
BeforeDetach	ConnectorLine	

“After” an Action Events

Event Name	Used By Object	Description
AfterPrint	Document	
	Diagram	
AfterChangeLayer	DiagramObject	
AfterEditCustomData	DiagramObject	
AfterGroup	DiagramObject	
AfterMove	DiagramObject	
AfterRotate	DiagramObject	
AfterSave	DiagramObject	
AfterSize	DiagramObject	
AfterUngroup	DiagramObject	
AfterAdjustmentMove	Shape	
AfterConnectorAttach	Shape	
AfterConnectorDetach	Shape	
AfterFontChange	Shape	
AfterStyleChange	Shape	
AfterTextChange	Shape	
AfterAttach	ConnectorLine	
AfterDetach	ConnectorLine	

Entity Events (iDiagrams)

Event Name	Used By Object	Description
EntitiesAbort	Document	
	Shape	
EntitiesFinished	Document	
	Shape	
EntitiesStart	Document	
	Shape	
EntitiesStep	Document	
EntityAccept	Shape	
EntityExecute	Shape	
EntityInitiate	Shape	
EntityLeave	Shape	
EntityStep	Shape	

Miscellaneous Events

Event Name	Used By Object	Description

GetInterface	Application
	Document
	Component
	Diagram
	DiagramObject
OutputWindowGoTo	Application
UserEvent	Application
	Document
	Component
	Diagram
	DiagramObject
FunctionValue	Document
ContextMenu	Diagram
	DiagramObject
LayerAdd	Diagram
LayerDelete	Diagram
LayerRename	Diagram
Load	DiagramObject
SetLink	Shape

Application Object

The Application object is the top of the iGrafx Professional object hierarchy. It is the primary interface to the iGrafx Professional object model. There can be only one iGrafx Professional Application object at a time running in your system.

The Application object contains:

- Access points to lower levels of the object hierarchy; for instance, Document, Diagram, DiagramObject, Shape, etc.
- Access to user interface objects, such as the grid, status bar, percent gauge, etc., that provide useful tools for use in writing VBA programs.
- Methods and events whose scope covers the entire application, and that provide application-level functions, such as delaying the execution of an action, or incorporating programming extensions.

Several application events are interesting in that they are the only way to do certain things. Listening to events at the application level, called Event Sink, takes a little doing (refer to the discussion of the ExtensionProject object), but is not difficult.

Properties, Methods, and Events

All of the Properties, methods, and events for the Application object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
ActiveDiagram	ActivateApplication	Activate
ActiveDocument	ArrangeIcons	BeforeKeyDown
ActiveMode	Cascade	BeforeWelcome
ActivePrinter	CloseAll	ChangeUnits
ActiveRouting	DoLater	Deactivate
ActiveUnits	ExecuteCommand	GetInterface
ActiveView	FireUserEvent	Move
Addins	Help	NewDocument
Application	Hint	OpenDocument
AsType	IsCommandAvailable	OutputWindowGoTo
Build	Maximize	Quit
Caption	Minimize	Resize
CommandBars	Output	Startup
CommandCategories	QuitApplication	UserEvent
Cursor	RegisterExtension	
DefaultFilePath	RegisterTimer	
DiagramTypes	RepaintAll	
Documents	Restore	
EventManager	TileHorizontal	
ExtensionProjects	TileVertical	
FontNames	UnregisterTimer	
FullName		
FullScreen		
Gallery		
GeometryHelper		
Grid		
Height		

Left
Name
OutputWindow
Parent
Path
PercentGauge
PopupWindows
RecentFiles
Ruler
SecurityLevel
ShapeLibraries
ShowFinished
StatusBar
Templates
Top
TrialVersion
UserCompany
UserName
VBE
Version
Visible
Width
Window
Windows
WindowState
Workspace

Activate Event

Syntax **Private Sub** *Application*_**Activate**()

Description The Activate event occurs when the iGrafx Professional application window is activated, or is brought to the front by user actions such as clicking on the application window, pressing ALT-TAB, or using the Task Manager. The Activate event fires when the application windows gains the focus.

You can use this event to perform actions when the application is activated, such as showing a floating form that appears when the application window gains the focus.

Example The following example shows how to set up the Application_Activate() event. The event executes every time the application window is activated. To try this example code, copy it all into a code window for a diagram. Press F5, and run "ConnectToAppEvent".

```
' Dimension an Application Object that hears events
' The "WithEvents" keyword switches on event monitoring
' This declaration is at the module level (not inside a Sub)
Public WithEvents AppMonitor As Application
' The main program
Public Sub ConnectToAppEvent()
    ' Create the Application Object
    ' Event monitoring was already enabled when AppMonitor was declared
    Set AppMonitor = Application
    ' Confirm the setup with a message
    MsgBox "The event is now active. Return to the diagram and try it."
End Sub
Private Sub AppMonitor_Activate()
    ' This code is what happens every time the Application is activated
    MsgBox "Applicaton activated."
End Sub
```

See Also ActivateApplication method

Deactivate event

Visible property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ActivateApplication Method

Syntax *Application*.**ActivateApplication**

Description The ActivateApplication method activates the application and brings it to the top of the Window list. This method is the same as if the application window is activated or brought to the front by user actions such as clicking on the application window, pressing ALT-TAB, or using the Task Manager.

The ActivateApplication method might be used after switching over to another application to return back to iGrafx Professional.

Example The following example uses the ActivateApplication method to return to iGrafx Professional from another application. First it launches WordPad, then returns back to iGrafx Professional. Note that the path name in the example finds Wordpad on Windows NT. If you are using a different operating system, change the path name accordingly.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxApp As Application
    ' Set the igxApp variable to the current Application object
    Set igxApp = Application.Application
    ' If WordPad isn't found, trap the error
    On Error GoTo ErrorHandler
    ' Launch WordPad
    RetVal = Shell("C:\Program Files\Windows NT\Accessories\WORDPAD.EXE", _
        vbMaximizedFocus)
    ' Return to iGrafx Professional
    Call igxApp.ActivateApplication
    MsgBox "WordPad was launched, but then we returned to iGrafx" _
        & Chr(13) & "Professional using the ActivateApplication method."
Exit Sub
' Do this if an error occurs
ErrorHandler:
    MsgBox "Couldn't find WordPad.exe. Please supply the" _
        & Chr(13) & "correct path to Wordpad.exe in the code."
End Sub
```

See Also Activate event

Deactivate event

Visible property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ActiveDiagram Property

Syntax *Application*.**ActiveDiagram**

Data Type Diagram object (read-only, See Object Properties)

Description The ActiveDiagram property returns the currently active Diagram object for the Application object. The currently active Diagram object is the diagram that currently has the focus (note that through automation, input could be provided to a diagram that isn't active).

This property provides the simplest way to operate on the current diagram. If there are no documents open, or the object that has the focus is not a diagram (such as when another Component object has the focus), then the ActiveDiagram property returns Nothing (for more information about object properties returning a Nothing value, refer to the Visual Basic programming documentation).

Example The following example uses the ActiveDiagram property of the Application object to display the name of the active Diagram.

```
' Dimension the variables
Dim igxApp As Application
Dim igxDiagram As Diagram
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxDiagram variable to the active Diagram object
Set igxDiagram = igxApp.ActiveDiagram
' Display the Name Property of the active diagram
MsgBox "The Active Diagram is called " & Chr(34) _
    & igxDiagram.Name & Chr(34)
```

See Also ActiveDocument property

Diagram object

iGrafx API Object Hierarchy

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ActiveDocument Property

Syntax *Application*.**ActiveDocument**

Data Type Document object (read-only, See Object Properties)

Description The ActiveDocument property returns the currently active Document object for the specified Application object. The currently active Document object is the document with a diagram or component that has the focus. This property provides the simplest way to operate on the current document.

A document does not really have a direct view: it is just a container for diagrams and other Component objects. A document can contain both diagrams and components. Components are such things as Scenarios (iGrafx Process) or Reports, but could be any data that there is a DLL (extension) that can handle it. These Component objects have to be stored at the document level.

The ActiveDiagram and ActiveDocument properties still return valid values even if the application has lost the focus. The ActiveDiagram property would fail (return Nothing) if the current view is of a component. If there are no documents open, then the ActiveDocument property returns Nothing.

Example The following example uses the ActiveDocument property of the Application object to display the name of the active document.

```
' Dimension the variables
Dim igxApp As Application
Dim igxDocument As Document
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxDocument variable to the active Document object
Set igxDocument = igxApp.ActiveDocument
' Display the Name Property of the active document
MsgBox "Active Document is " & Chr(34) & igxDocument.Name + Chr(34)
```

See Also ActiveDiagram property

Document object

iGrafx API Object Hierarchy

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ActiveMode Property

Syntax *Application*.**ActiveMode**

Data Type IxMode enumerated constant (read/write)

Description The ActiveMode property controls the current mode of the application. An application is always in one mode or another. Modes are states that are related to a particular tool, such as Draw mode, or Zoom mode.

The ActiveMode property is useful for macros that might require the Application to be in a certain mode for the actions to work. Could also write a Wizard or tutorial that walks a user through certain tasks, and switches the mode appropriately for the task to be learned or presented.

The IxMode constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixSelect
1	ixRotate
2	ixReshape
5	ixEditText
6	ixDrawShape
7	ixDrawLine
8	ixRenumber
9	ixDrawSquare
10	ixDrawRectangle
11	ixDrawPolygon
12	ixDrawCircle
13	ixDrawRoundedSquare
14	ixDrawRoundedRectangle
15	ixDrawSmoothPolygon
16	ixDrawEllipse
17	ixDrawPolyLine
18	ixDrawCurve
19	ixZoom
20	ixModeOther

Error Specifying an invalid enumerated Mode type produces an error (IGRAFX_E_INVALIDENUMERATEDTYPE). Use error trapping if your code could potentially supply the ActiveMode method with an invalid enumerated Mode type.

Example The following example uses the ActiveMode property to switch from one Mode to another, and then use the ActiveMode property to read the current mode.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Check and set the ActiveMode Property to Zoom
MsgBox "Click OK to switch to Zoom mode."
```

```
igxApp.ActiveMode = ixZoom
MsgBox "Click OK to switch to Edit Text mode."
igxApp.ActiveMode = ixEditText
MsgBox "Click OK to switch to Draw Shape mode."
igxApp.ActiveMode = ixDrawShape
' Display the ActiveMode Property of the Application
MsgBox "ActiveMode enumerator for Draw Shape mode is " & igxApp.ActiveMode
```

See Also ActiveRouting property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ActivePrinter Property

Syntax *Application*.**ActivePrinter**

Data Type String (read/write)

Description The ActivePrinter property specifies the current ActivePrinter name.

When you read the value of the ActivePrinter property, it returns the current printer and port. For example, it might return “HP LaserJet III on LPT2:”. When you set the value, the program uses a “loose matching” routine to set the active printer. For example, setting the ActivePrinter property to “HP Laser” or “LPT2” chooses “HP LaserJet III on LPT2:” if that is the printer on LPT2. If more than one printer matches the value you set, the first one alphabetically is used.

Example The following example uses the ActivePrinter property of the Application object to display the current active printer and port on the system.

```
' Dimension the variables
Dim igxApp As Application
Dim strPrinter As String
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Get the ActivePrinter property
strPrinter = igxApp.ActivePrinter
' Display the ActivePrinter property of the Application
MsgBox "The active printer is " & strPrinter
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ActiveRouting Property

Syntax *Application*.**ActiveRouting**

Data Type IxRouteType enumerated constant (read/write)

Description The ActiveRouting property specifies the active line routing tool to use for routing connector lines. The property provides the equivalent functionality of a user choosing a line routing type from the line tool dialog.

Setting this property puts the application into Draw Line mode (see the ActiveMode property, value 7, ixDrawLine).

The IxRouteType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixRouteDirect
1	ixRouteRightAngle
2	ixRouteCurved
3	ixRouteOrgChart
4	ixRouteCauseAndEffect
5	ixRouteLightningBolt
6	ixRouteCustom

Error Specifying an invalid enumerated Mode type produces an error (IGRAFX_E_INVALIDENUMERATEDTYPE). Use error trapping if your code could potentially supply the ActiveMode method with an invalid enumerated Mode type.

Example The following example uses the ActiveRoute property of the Application object to set and display the current active route enumerated value.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the ActiveRoute Property to ixRouteCurved
igxApp.ActiveRouting = ixRouteCurved
' Display the ActiveRoute Property of the Application
MsgBox "The active routing type enum is " & igxApp.ActiveRouting
```

See Also ActiveMode property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ActiveUnits Property

Syntax *Application*.**ActiveUnits**

Data Type IxUnits enumerated constant (read/write)

Description The ActiveUnits property specifies the type of units that are used in the toolbar and iGrafx Professional dialogs.

The IxUnits constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixUnitsInches
1	ixUnitsCentimeters

Example The following example uses the ActiveUnits property of the Application object to set and display the current active units type.

```
' Dimension the variables
Dim igxApp As Application
Dim strUnits As String
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Switch the type of units used
For Repeat = 1 To 4
   If igxApp.ActiveUnits = ixUnitsCentimeters Then
      ' Set the ActiveUnits property to inches
      igxApp.ActiveUnits = ixUnitsInches
      strUnits = "Inches"
   Else
      ' Set the ActiveUnits property to metric(centimeters)
      igxApp.ActiveUnits = ixUnitsCentimeters
      strUnits = "Centimeters"
   End If
   ' Display the units type of the Application
   MsgBox "The active units are " & strUnits
Next Repeat
```

See Also Ruler.Units property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ActiveView Property

Syntax *Application*.**ActiveView**

Data Type View object (read-only, See Object Properties)

Description The ActiveView property returns the currently active View object for the specified Application object. The currently active View object is the view that currently has the focus.

Be careful when using the View.Close method (also can be written as ActiveView.Close). When the view closes, Visual Basic closes it's associated code windows. You will not have access to the code window, and you may lose unsaved changes.

Example The following example uses the ActiveView to access the window width property of the view, and change the width of the view window.

```
' Dimension the variables
Dim igxApp As Application
Dim igxActiveView As View
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Retrieve the View object from the ActiveView Property
Set igxActiveView = igxApp.ActiveView
' Resize the view window using the ActiveView object
MsgBox "Click OK to resize the view window"
igxActiveView.Window.Width = 300
MsgBox "Click OK to continue"
```

See Also View object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

AddIns Property

Syntax *Application*.**AddIns**

Data Type Object

Description The AddIns property returns the collection of add-ins that have been registered for use with VBA. The AddIns property provides access to any installed add-ins. Add-ins are typically created using Visual Basic 6.0. The iGrafx Professional ExtensionProject object provides a similar functionality.

Example The following example demonstrates how to access the application's Addins collection, and display the number of Addins present in the system. Before trying this example, you need to reference the VBA Extensibility library. To do so, go to the Tools menu and select **Tools->References...** option. In the references list, look for "**Microsoft Visual Basic for Applications Extensibility 5.3**" and check it.

```
' Before trying this example, you need to reference the VBA Extensibility
' library. To do it, go to the Tools menu and select Tools->References...
' In the references list look for:
' "Microsoft Visual Basic for Applications Extensibility 5.3"
' and check it.
Sub main()
    ' Dimension the variables
    Dim igxAddins As VBIDE.AddIns
    ' Set our variable to the application addins collection
    Set igxAddins = Application.AddIns
    ' See if there are any addins present
    If igxAddins.Count = 0 Then
        MsgBox "There are no Addins registered."
    Else
        MsgBox "There are " & igsAddins.Count & " Addins registered."
    End If
End Sub
```

See Also RegisterExtension method

ExtensionProject object

ExtensionProjects object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Application Property

Syntax *Application*.**Application**

Data Type Application object (read-only, See Object Properties)

Description The Application property gives you access to the root object (Application) from any other object. “Application” is a property of every object in the iGrafx Professional API. The property is provided for congruency reasons only.

Example The following example retrieves the Application object of the application and displays it’s Caption Property.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Display the Caption Property of the Application
MsgBox "The application is " & igxApp.Caption
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ArrangeIcons Method

Syntax *Application*.**ArrangeIcons**

Description The ArrangeIcons method arranges the icons of minimized windows at the bottom of the iGrafx Professional window. This is the same as selecting Arrange Icons from the Window menu. If there are no windows minimized to icons, then this method has no effect.

Example The following example adds several new diagrams to the document, and minimizes all windows to icons. Then it shrinks the size of the main application window, which causes the icons to be lost off screen. Finally the ArrangeIcons method is invoked, which brings the icons back on screen, and in an orderly row.

```
' Dimension the variables
Dim igxApp As Application
Dim igxDiagram2 As Diagram
Dim igxDiagram3 As Diagram
Dim DocIndex As Integer
Dim WinIndex As Integer
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Create a new document in the application
MsgBox "Click OK to create 3 new diagrams."
igxApp.ActiveDocument.Diagrams.Add ("Diagram 2")
igxApp.ActiveDocument.Diagrams.Add ("Diagram 3")
igxApp.ActiveDocument.Diagrams.Add ("Diagram 4")
' Minimize all the documents
MsgBox "Click OK to minimize all diagrams to icons."
' Minimize the document window
igxApp.ActiveDocument.Windows.Item(1).WindowState = ixWindowMinimized
' Minimize all diagram windows
For DiagIndex = 1 To igxApp.ActiveDocument.Diagrams.Count
   With igxApp.ActiveDocument.Diagrams.Item(DiagIndex).Views.Item(1).Window
        .WindowState = ixWindowMinimized
   End With
Next DiagIndex
' Shrink the height of the main window to loose the icons
MsgBox "Click OK to shrink the main Application window"
igxApp.Window.Height = (igxApp.Window.Height / 2)
' Invoke the ArrangeIcons method
MsgBox "Click OK to Arrange the Icons, and bring them back on screen."
igxApp.ArrangeIcons
MsgBox "Click OK to continue."
```

See Also

Cascade method

Minimize method

TileHorizontal method

TileVertical method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

AsType Property

Syntax *Application*.**AsType**(*TypeName* As String) As Object

Data Type Object (read-only, See Object Properties)

Description The AsType property, together with the GetInterface event, provide an extendible type system for key iGrafx Professional objects.

When you extend an iGrafx Professional object using the GetInterface event, you need to keep in mind that other developers are using the event also. To be a good citizen, you should do the following:

- Be sure to pick a name that is likely to be unique for your AsType name. In the example, "Dinner" is too generic and it is possible that another developer could use the same name. Instead, follow the convention of using your name or your company name, a period, and a description of the type. For example, if you were writing a type that extended the Application to add additional internet capabilities, and your company name was "Micrografx", you could name your AsType name "Micrografx.InternetExtension".
- When you write code in the GetInterface event, keep it simple. You should not perform any time-consuming operation in the GetInterface event, such as querying a database or displaying a dialog box.
- When you write code in the GetInterface event, be aware of the current state of the Interface parameter. In the example, this is illustrated by the code fragment "Interface Is Nothing". If this code fragment evaluates to True, then it is safe to set the interface to your class. If this code fragment evaluates to False, then someone else has already responded to the event and set the interface to their class. If the latter condition arises, you should try changing your AsType name.

Example Using the AsType property, the GetInterface event, and VBA's support for Classes, you can extend key iGrafx Professional objects. The first step to doing this is creating a VBA class. The following example shows the creation of a simple class which has two properties—MainCourse, and Desert.

Insert a new class under ExtensionProject called Class1, and copy this block of code into it.

```
' Class
Public Property Get MainCourse() As String
    MainCourse = "Meatloaf"
End Property

Public Property Get Desert() As String
    Desert = "Cake"
End Property
```

These two blocks of code go in the ExtensionProject object's "This Application" code window.

```
' Run this to test the event
Sub Main()
   MsgBox "The main course is " & Application.AsType("Dinner").MainCourse
End Sub

' The GetInterface Event:
' The GetInterface event is fired whenever the AsType method is used.
' Based on the TypeName, redirect the interface to your custom class.
Private Sub Application_GetInterface(ByVal TypeName As String, Interface As
Object)
    ' If the broadcast type name is this, then set the interface
```

```
    If TypeName = "Dinner" Then
        ' TypeName gets broadcast everywhere, so check if
        ' something else grabbed and set the Interface first
        If Interface Is Nothing Then
            Set Interface = New Class1
        Else
            MsgBox "ERROR: Someone else is using the type Dinner"
        End If
    End If
End Sub
```

See Also GetInterface event

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

BeforeKeyDown Event

Syntax **Sub** *Application*_**BeforeKeyDown**(ByVal *KeyCode* As Integer, ByVal *Flags* As Long, *Cancel* As Boolean)

Description The BeforeKeyDown event occurs when a key is pressed on the keyboard. To prevent iGrafx Professional from seeing a particular key press, set the *Cancel* parameter to True.

To use Application events such as BeforeKeyDown, you need to declare an Application object using the "WithEvents" keyword. This declares an Application object with the event listening feature turned on.

The *KeyCode* parameter specifies the virtual-key code of the key being pressed. The following tables provides the integer key code values.

Key Codes

Name of Constant	Integer Value	Description
vbKeyLButton	1	Left mouse button
vbKeyRButton	2	Right mouse button
vbKeyCancel	3	CANCEL key
vbKeyMButton	4	Middle mouse button
vbKeyBack	8	BACKSPACE key
vbKeyTab	9	TAB key
vbKeyClear	12	CLEAR key
vbKeyReturn	13	ENTER key
vbKeyShift	16	SHIFT key
vbKeyControl	17	CTRL key
vbKeyMenu	18	MENU key
vbKeyPause	19	PAUSE key
vbKeyCapital	20	CAPS LOCK key
vbKeyEscape	27	ESC key
vbKeySpace	32	SPACEBAR key
vbKeyPageUp	33	PAGE UP key
vbKeyPageDown	34	PAGE DOWN key
vbKeyEnd	35	END key
vbKeyHome	36	HOME key
vbKeyLeft	37	LEFT ARROW key
vbKeyUp	38	UP ARROW key
vbKeyRight	39	RIGHT ARROW key
vbKeyDown	40	DOWN ARROW key
vbKeySelect	41	SELECT key
vbKeyPrint	42	PRINT SCREEN key
vbKeyExecute	43	EXECUTE key
vbKeySnapshot	44	SNAPSHOT key
vbKeyInsert	45	INS key
vbKeyDelete	46	DEL key
vbKeyHelp	47	HELP key
vbKeyNumlock	144	NUM LOCK key

Key A Through Key Z Are the Same as Their ASCII Equivalents: 'A' Through 'Z'

Name of Constant	Integer Value	Description
vbKeyA	65	A key
vbKeyB	66	B key
vbKeyC	67	C key
vbKeyD	68	D key
vbKeyE	69	E key
vbKeyF	70	F key
vbKeyG	71	G key
vbKeyH	72	H key
vbKeyI	73	I key
vbKeyJ	74	J key
vbKeyK	75	K key
vbKeyL	76	L key
vbKeyM	77	M key
vbKeyN	78	N key
vbKeyO	79	O key
vbKeyP	80	P key
vbKeyQ	81	Q key
vbKeyR	82	R key
vbKeyS	83	S key
vbKeyT	84	T key
vbKeyU	85	U key
vbKeyV	86	V key
vbKeyW	87	W key
vbKeyX	88	X key
vbKeyY	89	Y key
vbKeyZ	90	Z key

Key 0 Through Key 9 Are the Same as Their ASCII Equivalents: '0' Through '9'

Name of Constant	Integer Value	Description
vbKey0	48	0 key
vbKey1	49	1 key
vbKey2	50	2 key
vbKey3	51	3 key
vbKey4	52	4 key
vbKey5	53	5 key
vbKey6	54	6 key
vbKey7	55	7 key
vbKey8	56	8 key
vbKey9	57	9 key

Keys on the Numeric Keypad

Name of Constant	Integer Value	Description
vbKeyNumpad0	96	0 key
vbKeyNumpad1	97	1 key
vbKeyNumpad2	98	2 key
vbKeyNumpad3	99	3 key
vbKeyNumpad4	100	4 key
vbKeyNumpad5	101	5 key
vbKeyNumpad6	102	6 key
vbKeyNumpad7	103	7 key
vbKeyNumpad8	104	8 key
vbKeyNumpad9	105	9 key
vbKeyMultiply	106	MULTIPLICATION SIGN (*) key
vbKeyAdd	107	PLUS SIGN (+) key
vbKeySeparator	108	ENTER (keypad) key
vbKeySubtract	109	MINUS SIGN (-) key
vbKeyDecimal	110	DECIMAL POINT(.) key
vbKeyDivide	111	DIVISION SIGN (/) key

Function Keys

Name of Constant	Integer Value	Description
vbKeyF1	112	F1 key
vbKeyF2	113	F2 key
vbKeyF3	114	F3 key
vbKeyF4	115	F4 key
vbKeyF5	116	F5 key
vbKeyF6	117	F6 key
vbKeyF7	118	F7 key
vbKeyF8	119	F8 key
vbKeyF9	120	F9 key
vbKeyF10	121	F10 key
vbKeyF11	122	F11 key
vbKeyF12	123	F12 key
vbKeyF13	124	F13 key
vbKeyF14	125	F14 key
vbKeyF15	126	F15 key
vbKeyF16	127	F16 key

The *Flags* parameter specifies the repeat count, scan code, extended-key flag, context code, previous key-state flag, and transition-state flag, as shown in the following table.

Bits	Description
0–15	Specifies the repeat count for the current message. The value is the number of times the keystroke is auto-

	repeated as a result of the user holding down the key. If the keystroke is held long enough, multiple messages are sent. However, the repeat count is not cumulative.
16–23	Specifies the scan code. The value depends on the original equipment manufacturer (OEM).
24	Specifies whether the key is an extended key, such as the right-hand alt and ctrl keys that appear on an enhanced 101- or 102-key keyboard. The value is 1 if it is an extended key; otherwise, it is 0.
25–28	Reserved; do not use.
29	Specifies the context code. The value is 1 if the ALT key is down while the key is pressed; it is 0 if the WM_SYSKEYDOWN message is posted to the active window because no window has the keyboard focus.
30	Specifies the previous key state. The value is 1 if the key is down before the message is sent, or it is 0 if the key is up.
31	Specifies the transition state. The value is always 0 for a WM_SYSKEYDOWN message.

Example The following example sets up the BeforeKeyDown event.

```
' Dimension an Application Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of Application objects. This declaration is at the module level
' (not inside a Sub).
Public WithEvents AppMonitor As Application

' The main program. Run this Sub to establish the event
Public Sub ConnectToAppEvent()
    ' Create the Application Object
    ' Event monitoring was already enabled when AppMonitor was declared
   Set AppMonitor = Application
   ' Confirm the setup with a message
   MsgBox "The event is now active. Return to the diagram and try it."
End Sub

Private Sub AppMonitor_BeforeKeyDown(ByVal KeyCode As Integer, ByVal Flags As
Long, Cancel As Boolean)
   ' Sound a bell every time a key is pressed
   Beep
End Sub
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

BeforeWelcome Event

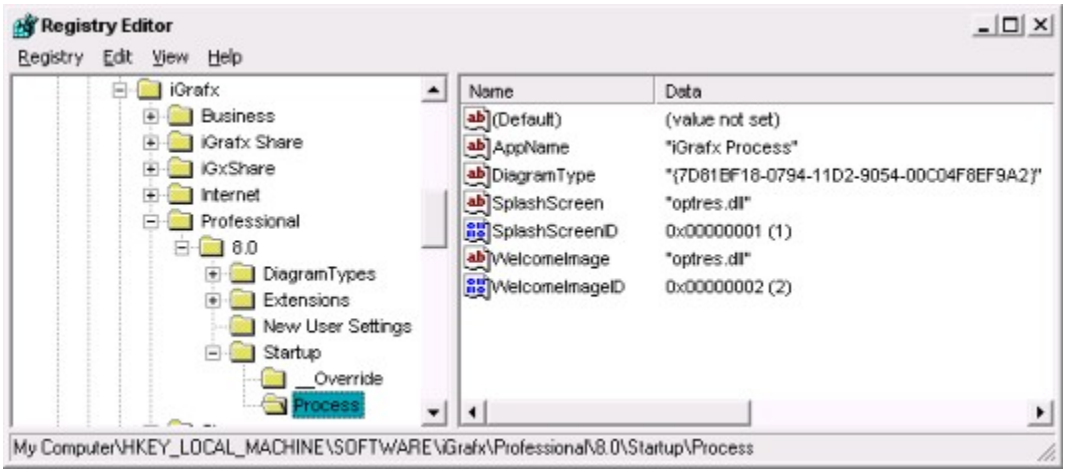
Syntax **Sub** *Application*_**BeforeWelcome**(*StartupString* As String, *CancelWelcome* As Boolean)

Description The BeforeWelcomeEvent occurs before the iGrafx Professional Welcome screen is displayed. This event gives the programmer an opportunity to display a custom welcome dialog. It is not recommended that you perform other actions in this event, as they will not occur if the user has chosen to not display the welcome dialog. To prevent iGrafx Professional from displaying the default Welcome screen, set the *Cancel* parameter to True.

The *StartupString* parameter contains the startup string specified by the shortcut that launched the application. For example, if the executable is launched with "startup=process" the *StartupString* parameter contains the string "process".

You can define your own startup string; for example "startup=myapp". In the registry, you can register an alternate splash screen, an alternate welcome dialog graphic, a default diagram type, and a different application name.

The following illustration shows the registry startup settings for "startup=process". The registry key "Process" is defined in HKEY_LOCAL_MACHINE\Software\iGrafx\Professional\8.0\Startup.



The following definitions are applicable to the entries on the right side of the illustration. Note that all the values for these entries are optional.

- **AppName** defines what will show in the caption of the application window.
- **DiagramType** specifies the default DiagramType (if the user has turned off the welcome dialog).
- **SplashScreen** specifies where the GIF or BMP is located for the splash screen. It can be in a DLL, or it may be a plain BMP or GIF file.
- **SplashScreenID** specified the resource ID if the bitmap (or GIF) is in a DLL. If this is not specified, iGrafx Professional will load the DLL and try to load in the first 5 resource IDs until it finds a GIF or BMP. For a GIF file, the resource type must be "GIF".
- **WelcomeImage** specifies where the GIF or BMP is located (it can be in a DLL).
- **WelcomeImageID** specifies the resource ID if the bitmap (or GIF) is in a DLL.

Example The following example shows a custom welcome dialog in place of the standard Welcome dialog. To try this example, first create an Extension Project through the Extension Projects dialog in the user interface:

Tools->Visual Basic->Extension Projects…->New

Next, copy the following block of code into the "This Application" area of the ExtensionProject.

You find this in the Visual Basic Project Explorer, under the name you gave to your new Extension Project. Once this is done and saved, exit iGrafx Professional and restart it. The custom welcome dialog should appear rather than the standard iGrafx Professional Welcome dialog.

```
Private Sub Application_BeforeWelcome(StartupString As String, _
    Cancel As Boolean)
    If (MsgBox("My custom welcome screen." _
    & Chr(13) & "Create a new document?", _
    vbYesNo)) = vbYes Then
        Application.Documents.New (1)
    End If
    Cancel = True
End Sub
```

See Also Startup event

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Build Property

Syntax *Application*.**Build**

Data Type String (read-only)

Description The Build property returns the iGrafx Professional build number. This is useful to insure that a program written for one build cannot be run on any other versions. If a section of your code works with a particular build of iGrafx Professional, but not with other builds, you can check this property before entering that section of code.

Example The following example retrieves the build number from the Build property of the application and displays the value in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
Dim strBldNumber As String
' Set the ixappApp variable to the current Application object
Set igxApp = Application.Application
' Gets the build number from the application
strBldNumber = igxApp.Build
' Displays the build number in a Message Box
MsgBox "Application Name: " & Application.Caption & Chr(13) _
    & "Build Number: " & strBldNumber
' Change the Caption name
igxApp.Caption = "My Application"
```

See Also Name property

Version property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Cascade Method

Syntax *Application*.**Cascade**

Description The Cascade method arranges all open windows within the iGrafx Professional application window in a cascaded fashion. The windows are arranged so the title bar of each window is visible. The method works with one or more windows, and provides the same functionality as the Cascade option on the Window menu.

Example The following example invokes the Cascade method of the Application object to arrange all open windows in a cascade manner.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Create a new document in the application
igxApp.Documents.New
MsgBox "New document added. Click OK to Cascade the windows."
' Invokes the cascade method for arranging the open windows
igxApp.Cascade
MsgBox "All windows have been Cascaded."
```

See Also ArrangeIcons method

TileHorizontal method

TileVertical method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ChangeUnits Event

Syntax **Sub** *Application*_**ChangeUnits**

Description The ChangeUnits event occurs when the user changes units (from inches to centimeters or from centimeters to inches). If you have shapes that are displaying their size, for example, you would want to use this event to update the display to match the user's current units selection.

Example The following example sets up the ChangeUnits event. When the units are changed, a message box is displayed showing the active units.

```
' Set a reference to an Application Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of Application objects. This declaration is at the module level
' (not inside a Sub)
Public WithEvents AppMonitor As Application

' The main program. Run this Sub to establish the event
Public Sub ConnectToAppEvent()
    ' Create the Application Object
    ' Event monitoring was already enabled when AppMonitor was declared
    Set AppMonitor = Application
    ' Let's confirm the setup with a message
    MsgBox "The event is now active. Return to the diagram and try it."
End Sub

' Procedure that executes when the units are changed
Private Sub AppMonitor_ChangeUnits()
    ' Check which units are active and display it
    If Application.ActiveUnits = ixUnitsCentimeters Then
        MsgBox "The Units are now centimeters."
    Else
        MsgBox "The Units are now inches."
    End If
End Sub
```

See Also ActiveUnits property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

CloseAll Method

Syntax *Application*.**CloseAll**

Description The CloseAll method closes all of the open windows within the iGrafx Professional application.

Caution This method does not prompt the user to save any unsaved work. It closes all open diagrams, documents, etc., without performing a Save operation.

Example The following example invokes the CloseAll method of the Application object to close all open windows without saving. Warning: this code example closes all open documents without saving. You will lose any information in the application, including information in the Visual Basic editor that is associated with the documents.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Create a new document in the application
igxApp.Documents.New
' Invokes the CloseAll Method for closing all of the open windows
igxApp.CloseAll
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

CommandBars Property

Syntax *Application*.**CommandBars**

Data Type CommandBars collection object (read-only, See Object Properties)

Description The CommandBars property returns the CommandBars collection for the application, which represents the menu bar and all the toolbars in iGrafx Professional. By obtaining the collection of CommandBar objects, the programmer can add to, remove from, or modify the toolbars and menus.

The CommandBars object appears at three levels, Application, Document, and Diagram. You can use the CommandBars collection to customize command bars separately at each level.

Example The following example retrieves the number of CommandBar objects, and displays this number in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
Dim igxCommandBars As CommandBars
Dim lCount As Long
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxCommandBar variable to the CommandBars collection
Set igxCommandBars = igxApp.CommandBars
' Retrieve the number of CommandBar objects
lCount = igxCommandBars.Count
' Display the number of CommandBar objects
MsgBox "The number of CommandBar objects is " & lCount
```

See Also CommandCategories property

CommandBar object

CommandBars object

CommandBarItem object

CommandBarItems object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

CommandCategories Property

Syntax *Application*.**CommandCategories**

Data Type CommandCategories collection object (read-only, See Object Properties)

Description The CommandCategories property returns the CommandCategories collection object for the application, which represents the command categories listed in the Customize dialog box.

Example The following example retrieves the number of CommandCategory objects and displays this number in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
Dim igxCommandCategories As CommandCategories
Dim lCount As Long
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxCommandCategories variable to the CommandCategories collection
Set igxCommandCategories = igxApp.CommandCategories
' Retrieve the number of CommandCategory objects
lCount = igxCommandCategories.Count
' Display the number of CommandCategory objects
MsgBox "The number of CommandCategory objects is " & lCount
```

See Also

CommandBars property

CommandCategory object

CommandCategories object

CommandBar object

CommandBarItem object

CommandBarItems object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Cursor Property

Syntax *Application*.**Cursor**

Data Type Cursor object (read-only, See Object Properties)

Description The Cursor property returns the Cursor object for the application. The Cursor object represents the cursor being used by iGrafx Professional. The Type property of the Cursor object sets the current cursor type (by means of the IxCursor constant). Setting the IxCursor constant controls the currently displayed cursor in the application.

Example The following example sets a cursor object, and then reads its X, Y, and Type values.

```
' Dimension the variables
Dim igxApp As Application
Dim igxCursor As Cursor
' Set the igxApp variable to the current Application object.
Set igxApp = Application.Application
' Set the igxCursor variable to the Cursor object for the app.
Set igxCursor = igxApp.Cursor
' Display the current X coordinate of the cursor.
With igxCursor
    MsgBox "Attributes of the cursor: " + Chr(13) + Chr(13) + _
    "X Position: " & .XPosition & Chr(13) & _
    "Y Position: " & .YPosition & Chr(13) & _
    "Type: " & .Type
End With
```

See Also Cursor object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Deactivate Event

Syntax **Sub** *Application*_**Deactivate**()

Description The Deactivate event occurs when the application window is deactivated, or is placed in the background by user actions such as clicking on another application's window, pressing ALT-TAB, or using the Task Manager (that is, iGrafx Professional loses the focus). Use this event to perform actions when the application is deactivated.

Example The following example sounds the system bell sound every time iGrafx Professional is deactivated.

```
' Dimension an Application Object that hears events
' The "WithEvents" keyword switches on event monitoring
' This declaration is at the module level (not inside a Sub)
Public WithEvents AppMonitor As Application
' The main program
Public Sub ConnectToAppEvent()
    ' Create the Application Object
    ' Event monitoring was already enabled when AppMonitor was declared
    Set AppMonitor = Application
    ' Confirm the setup with a message
    MsgBox "The event is now active. Return to the diagram and try it."
End Sub
Private Sub AppMonitor_Deactivate()
    ' This code is what happens every time the Application is activated
    Bell
End Sub
```

See Also Activate event

ActivateApplication method

Visible property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

DefaultFilePath Property

Syntax *Application*.**DefaultFilePath**

Data Type String (read/write)

Description The DefaultFilePath property specifies the path that is used as the default location for the Open File or Save File dialogs. A valid value is any string that corresponds to the operating system's directory and file naming conventions.

Example The following example retrieves the default file path of the application for saving files, and displays this path in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
Dim strFilePath As String
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the strFilePath variable to the default file path
strFilePath = igxApp.DefaultFilePath
' Display the default file path for saving
MsgBox "The default file path for saving is " & strFilePath
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

DiagramTypes Property

Syntax *Application*.**DiagramTypes**

Data Type DiagramTypes collection object (read-only, See Object Properties)

Description The DiagramTypes property returns the DiagramTypes collection for the Application object. The DiagramTypes collection is used to access any of the installed diagram types in the application.

Example The following example retrieves the number of DiagramType objects that are in the DiagramTypes collection, and displays this number in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
Dim igxDiagramTypes As DiagramTypes
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxDiagramTypes variable to the DiagramTypes collection
Set igxDiagramTypes = igxApp.DiagramTypes
' Display the number of DiagramType objects in the collection
MsgBox "The number of DiagramType objects is " & igxDiagramTypes.Count
```

See Also DiagramType object

DiagramTypes object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Documents Property

Syntax *Application*.**Documents**

Data Type Documents collection object (read-only, See Object Properties)

Description The Documents property returns the Documents collection for the Application object. The Documents collection is used to access the individual open documents in the application.

Example The following example retrieves the number of Document objects that are in the Documents collection, and displays this number in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
Dim igxDocuments As Documents
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the strFilePath variable to the default file path
Set igxDocuments = igxApp.Documents
' Display the number of Document objects in the collection
MsgBox "The number of Document objects is " & igxDocuments.Count
```

See Also

ActiveDiagram property

ActiveDocument property

Document object

Documents object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

DoLater Method

Syntax *Application*.**DoLater**(*Callback* As Callback)

Description The DoLater method allows you to schedule or postpone certain actions (as coded in a "Callback" class) until all other system tasks are finished. The *Callback* parameter specifies a VBA class that implements the Callback object. The Callback object has one method that you must implement, called Execute.

The DoLater method calls the Execute procedure in your "Callback" VBA class as soon as the application has some idle time (e.g. after the application finishes processing all the messages that are currently in the queue).

You might use DoLater to execute some code after some operation completes, for example you might want to look at a diagram after a series of deletions. By posting your callback to the message queue using DoLater, you can look at the diagram after the operation in progress has completed.

Example The following example sets up a class that has implemented Callback. Then it shows how to use DoLater with that class.

Create a new class called Class1 under a diagram project. Copy this first block of code into the new class.

```
' Class1
' Force explicit variable declarations (corrects scope)
Option Explicit
' Need to specify that this class uses callback
Implements Callback

Private Sub Callback_Execute()
    MsgBox "Finished deleting objects"
End Sub
```

Copy this second block of code into the diagram project, and run it.

```
Public Sub Test()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim X, Y As Integer
    ' Dimension variable as our callback class
    Dim igxFinished As New Class1
    ' Create an instance of our callback class
    MsgBox "Click OK to create 10 shapes"
    ' Create some shapes
    For X = 1 To 10
        For Y = 1 To 10
            If X + Y = 2 Then Application.DoLater igxFinished
            Application.ActiveDiagram.DiagramObjects.AddShape _
                1500 * X, 1440 * Y
        Next Y
    Next X
End Sub
```

See Also Callback object

DoAfterCurrentChangeBlock event

DoAfterTopChangeBlock event

RegisterTimer method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

EventManager Property

Syntax *Application*.**EventManager**

Data Type EventManager object (read-only, See Object Properties)

Description The EventManager property returns the EventManager object for the Application object. This object has one property, CancelBubble. If you set the EventManager.CancelBubble property to True, you cancel further bubbling of the currently executing event.

For example, when you double click on a shape, that event bubbles through multiple controls. For a shape, the number of controls the event bubbles through is effectively doubled if the event is an "Extender" event. For more information on extenders, see the DiagramObject object.

The following list specifies the controls (in order of arrival) that an extender event would travel through.

- The VBA control for the shape (if there is one created).
- The AnyShape control at the Diagram level.
- The AnyObject control at the Diagram level.
- The AnyShape control at the Document level.
- The AnyObject control at the Document level.
- The AnyShape control at the Document's DiagramType level.
- The AnyObject control at the Document's DiagramType level.
- The ShapeClass's shape control at the Document level.
- The Application's DiagramType AnyShape control.
- The Application's DiagramType AnyObject control.
- Any Extension projects AnyShape control
- Any Extension projects AnyObject control

If you were to set CancelBubble = True in the event handler in the VBA control for the shape, the event would stop there and would not go to all the other controls.

For more information, see Event Bubbling in the iGrafx System Developer's Guide.

Errors If there is no current event bubble to cancel, an error is produced (IGRAFX_E_NOCURRENTEVENT). Use error trapping to handle an error due to the absence of an event bubble.

Example The following example sets up a Shape_BeforeClick event at the Diagram level, and also at the Document level. When the user clicks a shape, both events are fired, one after the other. However, if the user chooses to cancel the bubble, the Document-level event never gets fired.

To try this example, put each of these events in the code window indicated. Add a shape to the diagram if necessary, and click any shape.

```
' This is the Diagram level event
' Put this in a Diagram code window
Private Sub AnyShape_BeforeClick(ByVal X As Double, ByVal Y As Double,
Cancel As Boolean)
    ' Ask the user if they wish to cancel the event bubble
    If MsgBox("Diagram level event fired. Cancel bubble?", vbYesNo) _
    = vbYes Then
        EventManager.CancelBubble = True
    Else
```

```
        EventManager.CancelBubble = False
    End If
End Sub

-------------------------------------------------------

' This is the Document level event
' Put this in the ThisDocument code window
Private Sub AnyShape_BeforeClick(ByVal X As Double, ByVal Y As Double,
Cancel As Boolean)
    MsgBox "Document level event fired."
End Sub
```

See Also EventManager object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ExecuteCommand Method

Syntax *Application*.**ExecuteCommand** *Command* As IxBuiltInCommand

Description The ExecuteCommand method allows the programmer to execute any of the built in iGrafx Professional commands.

The IxBuiltInCommand constant defines the valid values for the *Command* argument, and are listed in the following table.

Value	Name of Constant
53249	ixFileNew
53250	ixFileOpen
53251	ixFileClose
53252	ixFileCloseAll
53253	ixFileSave
53254	ixFileSaveAs
53255	ixFileSaveAll
53256	ixFileSaveWorkspace
53257	ixFileSaveTemplate
53258	ixFilePageSetup
53259	ixFilePrintPreview
53260	ixFilePrint
53261	ixFilePrintSetup
53262	ixFileSendMail
53263	ixEditUndo
53264	ixEditRedo
53265	ixEditCut
53266	ixEditCopy
53267	ixEditPaste
53268	ixEditPasteSpecial
53269	ixEditClear
53270	ixEditDuplicate
53271	ixEditSelectAll
53272	ixEditSelect
53273	ixEditFind
53274	ixOleInsertNew
53275	ixOleEditLinks
53276	ixInsertPictureFromFile
53277	ixExportDiagram
53281	ixZoomComboBox
53282	ixZoomIn
53283	ixZoomOut
53284	ixZoomTool
53285	ixViewZoom
53287	ixViewFullScreen
53288	ixViewToolbar
53289	ixViewStatusBar
53290	ixViewRulers

53294	ixShowGuidelines
53295	ixShowGrid
53299	ixAddVerticalGuideline
53300	ixAddHorizontalGuideline
53301	ixMeasureInches
53302	ixMeasureCentimeters
53304	ixFormatDiagram
53305	ixFormatShapeNumbers
53306	ixFormatShapeNumberOn
53307	ixProperties
53308	ixFillComboBtn
53309	ixPatternComboBtn
53310	ixLineColorComboBtn
53311	ixLineWeightComboBtn
53312	ixLineStyleComboBtn
53313	ixArrowheadsComboBtn
53314	ixCrossoversComboBtn
53315	ixSourceArrowComboBtn
53316	ixDestArrowComboBtn
53317	ixShadowComboBtn
53318	ix3dPopup
53319	ixFormatPainter
53320	ixShowNodes
53321	ixFormatBold
53322	ixFormatItalic
53323	ixFormatUnderline
53324	ixFormatStrikethrough
53325	ixFormatOpaqueText
53326	ixFormatFitToText
53327	ixTextColorComboBtn
53328	ixTextAlignVertComboBtn
53329	ixTextAlignHorzComboBtn
53330	ixFontNameComboBox
53331	ixFontSizeComboBox
53332	ixFormatFont
53333	ixAlignTextLeft
53334	ixAlignTextHCenter
53335	ixAlignTextRight
53336	ixAlignTextTop
53337	ixAlignTextMiddle
53338	ixAlignTextBottom
53339	ixVerticalText
53340	ixBullets
53341	ixIndentLeft
53342	ixIndentRight

53343	ixSpacingIncrease
53344	ixSpacingDecrease
53345	ixConnectShapes
53346	ixArrangeBringToFront
53347	ixArrangeSendToBack
53348	ixArrangeBringForward
53349	ixArrangeSendBackward
53350	ixRotateByAngle
53351	ixArrangeRotateRight
53352	ixArrangeRotateLeft
53353	ixArrangeFlipHorizontal
53354	ixArrangeFlipVertical
53355	ixArrangeGroup
53356	ixArrangeUngroup
53357	ixArrangeAlignLeft
53358	ixArrangeAlignHCenter
53359	ixArrangeAlignRight
53360	ixArrangeAlignTop
53361	ixArrangeAlignVCenter
53362	ixArrangeAlignBottom
53363	ixEvenSpacingHCenters
53364	ixEvenSpacingVCenters
53365	ixEvenSpacingHEdges
53366	ixEvenSpacingVEdges
53367	ixMakeSameSizeWidth
53368	ixMakeSameSizeHeight
53369	ixMakeSameSizeBoth
53370	ixMakeSameSizeFitToText
53371	ixConvertToShape
53372	ixConvertToGraphic
53373	ixCombineConnectOpen
53374	ixCombineConnectClosed
53375	ixCombineDisconnect
53376	ixCombineJoin
53377	ixCombineIntersect
53381	ixCombineOutline
53382	ixCombineSlice
53383	ixReplaceShape
53384	ixArrangeReverseEnds
53386	ixLayerManager
53387	ixLayerAddNew
53388	ixLayerEditAll
53389	ixLayerMoveTo
53390	ixLayerMoveBack
53391	ixLayerMoveForward
53392	ixLink

53393	ixInsertSPCDiagram
53394	ixToolsSpelling
53395	ixToolsProtectDiagram
53398	ixExportSelected
53399	ixToolsCustomize
53400	ixToolsOptions
53401	ixRunDesigner
53402	ixRunPicturePublisher
53403	ixSelectorTool
53404	ixRotateTool
53406	ixZoomPopup
53407	ixShapeTool
53408	ixConnectorLinesComboButton
53409	ixLineRouteDirect
53410	ixLineRouteRightAngle
53411	ixLineRouteCurved
53412	ixLineRouteOrgChart
53413	ixLineRouteCauseAndEffect
53414	ixDrawToolPopup
53415	ixDrawToolSquare
53416	ixDrawToolRoundedSquare
53417	ixDrawToolRectangle
53418	ixDrawToolRoundedRectangle
53419	ixDrawToolCircle
53420	ixDrawToolEllipse
53421	ixDrawToolPolygon
53422	ixDrawToolSmoothedPolygon
53423	ixLineRoutePolyline
53424	ixLineRoutePolyBezier
53425	ixTextTool
53426	ixRenumberTool
53427	ixDataSetupFields
53428	ixDataOptions
53429	ixDataInputReport
53430	ixDataOutputReport
53431	ixDataMetricsReport
53432	ixDataUpdate
53437	ixWindowNewWindow
53438	ixWindowTileHorzontal
53439	ixWindowSply
53440	ixContextHelp
53441	ixHelpIndex
53442	ixHelpOfficeCompatible
53443	ixHelpAbout
53444	ixShapeStyleComboBox
53445	ixStyleAddShapeStyle

53446	ixStyleEditShapeStyle
53447	ixLineStyleComboBox
53448	ixStyleAddLineStyle
53449	ixStyleEditLineStyle
53450	ixTextStyleComboBox
53451	ixStyleAddTextStyle
53452	ixStyleEditTextStyle
53457	ixViewEntityMgr
53458	ixActiveDiagramOptions
53459	ixActiveDiagramExecute
53460	ixActiveDiagramStop
53461	ixActiveDiagramPause
53462	ixActiveDiagramStep
53463	ixToolsShowIDE
53464	ixToolsDesignMode
53465	ixVBAShowCode
53466	ixVBAShowProperties
53467	ixVBAControls
53468	ixVBAInsertCheckBox
53469	ixVBAInsertTextBox
53470	ixVBAInsertCommandButton
53471	ixVBAInsertOptionButton
53472	ixVBAInsertListBox
53473	ixVBAInsertComboBox
53474	ixVBAInsertToggleButton
53475	ixVBAInsertSpinButton
53476	ixVBAInsertScrollBar
53477	ixVBAInsertLabel
53478	ixVBAInsertImage
53479	ixVBAMoreControls
53480	ixViewPageBreaks
53481	ixPageUp
53482	ixPageDown
53483	ixFillColorControl
53484	ixLineColorControl
53485	ixTextColorControl

Example The following example invokes the ExecuteCommand method to zoom in and out on the view.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Invoke the ExecuteCommand method to do Zoom in and out
MsgBox "Click OK to zoom in"
igxApp.ExecuteCommand (ixZoomIn)
```

```
MsgBox "Click OK to zoom out"
igxApp.ExecuteCommand (ixZoomOut)
MsgBox "Click OK to continue"
```

See Also IsCommandAvailable method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ExtensionProjects Property

Syntax *Application*.**ExtensionProjects**

Data Type ExtensionProjects collection object (read-only, See Object Properties)

Description The ExtensionProjects property returns the ExtensionProjects collection for the application. For more information, refer to the documentation of the ExtensionProject object.

Example The following example retrieves the number of ExtensionProject objects that are in the ExtensionProjects collection, and displays this number in a message box.

```
' Dimension the variables
Dim igxApp As Application
Dim igxExtensionProjects As ExtensionProjects
' Set the ixappApp variable to the current Application object
Set igxApp = Application.Application
' Set igxExtensionProjects to the ExtensionProjects collection
Set igxExtensionProjects = igxApp.ExtensionProjects
' Display the number of ExtensionProject objects in the collection
MsgBox "The number of ExtensionProject objects in the collection is " _
    & igxExtensionProjects.Count
```

See Also ExtensionProject object

ExtensionProjects object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

FireUserEvent Method

Syntax *Application*.**FireUserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The FireUserEvent method fires the specified "UserEvent" event on the Application object. You can use this functionality to send messages to any event sinks listening to Application-level events.

You must pick an event identifier string to use for your event. You might choose to use something like your company name followed by the event name. You should choose a name that won't conflict with names picked by other developers.

You can pass one parameter to the event. This parameter is a variant, so one logical choice is to pass a VBA class. A Variant variable can be assigned to a VBA class. This situation allows the programmer to pass any type of object or variable back and forth between the FireUserEvent method and the UserEvent event.

Then, you can write code in a UserEvent event handler to do something when your event fires. This code should be of the form:

```
If EventIdentifier = "<<Your identifier string>>" Then
   << Write your code here >>
End If
```

Example The following example defines a new user event called "ShowUsers". The parameter that gets passed is a class, which has one property called Count. The event handler displays the parameter's Count property.

The following code defines a simple class with one property. Create a new class in the Document project called Class1 and copy this code into it.

```
' Class1
' It contains one property, read only
Public Property Get Count() As Long
   Count = 25
End Property
```

The following code is the main program. Copy this, and the UserEvent subroutine, into the Diagram project.

```
' Run this subroutine to test the event
Public Sub Main()
   ' Fire the UserEvent
   Application.FireUserEvent "ShowUsers", New Class1
End Sub

' This event handler runs every time any FireUserEvent method
' is used in the system
Private Sub Application_UserEvent(ByVal EventIdentifier As String, ByVal
Parameter As Variant)
   ' Check if the Identifier string is the one we want
   If EventIdentifier = "ShowUsers" Then
       ' Redirect to Class1
       MsgBox "The number of users is " & Parameter.Count
   End If
End Sub
```

See Also UserEvent event

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

FontNames Property

Syntax *Application*.**FontNames**

Data Type FontNames collection object (read-only, See Object Properties)

Description The FontNames property returns the FontNames collection of the Application object. The FontNames collection is used to access all of fonts that are available to iGrafx Professional.

Example The following example retrieves the number of font names that are in the FontNames collection, and displays this number in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
Dim igxFontNames As FontNames
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxFontNames variable to the FontNames collection
Set igxFontNames = igxApp.FontNames
' Display the number of font names in the collection
MsgBox "The number of font names in the collection is " & igxFontNames.Count
```

See Also Font object

FontNames object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

FullScreen Property

Syntax *Application*.**FullScreen**[= {**True | False**}]

Data Type Boolean (read/write)

Description The FullScreen property specifies whether iGrafx Professional is in FullScreen mode or Window mode. The property only affects windows that contain diagrams, and causes all the interface elements to be hidden. Windows that contain reports or scenarios (Components) do not respond to the FullScreen property.

Example The following example switches the application to full screen mode, and then back to window mode. The currently active window is displayed on the full screen.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set FullScreen Property to True
MsgBox "Click OK for full screen mode"
igxApp.FullScreen = True
MsgBox "Click OK for window mode"
igxApp.FullScreen = False
MsgBox "Click OK to continue"
```

See Also Window object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Gallery Property

Syntax *Application*.**Gallery**

Data Type Gallery object (read-only, See Object Properties)

Description The Gallery property returns the Gallery object for the Application object. Through the Gallery object, you have access to properties and methods that control how the gallery is displayed, which tab is active, which sub-pane is displayed, etc. For example, you could use the Gallery property to dock the gallery on the left side of the screen and to activate the Font page.

Example The following example gets the application's Gallery object, and then use the GalleryPane.Activate method to change the active pane being displayed in the gallery.

```
' Dimension the variables
Dim igxApp As Application
Dim igxGallery As Gallery
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Get the application's Gallery object
Set igxGallery = igxApp.Gallery
' Activate some of the gallery panes
igxGallery.GalleryPanes.Item(1).Activate
MsgBox "The Fill Color Gallery"
igxGallery.GalleryPanes.Item(2).Activate
MsgBox "The Font Gallery"
igxGallery.GalleryPanes.Item(3).Activate
MsgBox "The Line Style Gallery"
igxGallery.GalleryPanes.Item(4).Activate
MsgBox "The Arrow Style Gallery"
```

See Also Gallery object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

GeometryHelper Property

Syntax *Application*.**GeometryHelper**

Data Type GeometryHelper object (read-only, See Object Properties)

Description The GeometryHelper property returns a GeometryHelper object. This is an application-level object that contains a set of geometry functions that the programmer can use in the development of iShapes™.

For code examples, refer to documentation of the GeometryHelper object.

See Also GeometryHelper object

GraphicBuilder object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

GetInterface Event

Syntax **Sub** *Application*_**GetInterface**(*TypeName* As String, *Interface* As Object)

Description The AsType property, together with the GetInterface event, provide an extendible type system for key iGrafx Professional objects.

When you extend an iGrafx Professional object using the GetInterface event, you need to keep in mind that other developers are using the event also. To be a good citizen, you should do the following:

- Be sure to pick a name that is likely to be unique for your AsType name. In the example, "Dinner" is too generic and it is possible that another developer could use the same name. Instead, follow the convention of using your name or your company name, a period, and a description of the type. For example, if you were writing a type that extended the Application to add additional internet capabilities, and your company name was "Micrografx", you could name your AsType name "Micrografx.InternetExtension".
- When you write code in the GetInterface event, keep it simple. You should not perform any time-consuming operation in the GetInterface event, such as querying a database or displaying a dialog box.
- When you write code in the GetInterface event, be aware of the current state of the Interface parameter. In the example, this is illustrated by the code fragment "Interface Is Nothing". If this code fragment evaluates to True, then it is safe to set the interface to your class. If this code fragment evaluates to False, then someone else has already responded to the event and set the interface to their class. If the latter condition arises, you should try changing your AsType name.

Example Using the AsType property, the GetInterface event, and VBA's support for Classes, you can extend key iGrafx Professional objects. The first step to doing this is creating a VBA class. The following example shows the creation of a simple class which has two properties—MainCourse, and Desert.

Insert a new class under ExtensionProject called Class1, and copy this block of code into it.

```
' Class
Public Property Get MainCourse() As String
    MainCourse = "Meatloaf"
End Property

Public Property Get Desert() As String
    Desert = "Cake"
End Property
```

These two blocks of code go in the ExtensionProject object's "This Application" code window.

```
' Run this to test the event
Sub Main()
   MsgBox "The main course is " & Application.AsType("Dinner").MainCourse
End Sub

' The GetInterface Event:
' The GetInterface event is fired whenever the AsType method is used.
' Based on the TypeName, redirect the interface to your custom class.
Private Sub Application_GetInterface(ByVal TypeName As String, Interface As
Object)
    ' If the broadcast type name is this, then set the interface
    If TypeName = "Dinner" Then
        ' TypeName gets broadcast everywhere, so we need to check if
```

```
            ' something else grabbed and set the Interface first
            If Interface Is Nothing Then
                Set Interface = New Class1
            Else
                MsgBox "ERROR: Someone else is using the type Dinner"
            End If
        End If
    End Sub
```

See Also AsType property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Grid Property

Syntax *Application*.**Grid**

Data Type Grid object (read-only, See Object Properties)

Description The Grid property returns the Grid object for the Application object. The Grid object allows the programmer to control grid and grid snapping options. There is only one grid object, and it is used for all currently open documents. Creating multiple grid objects in the Visual Basic environment is allowed, but serves little purpose since all grid objects affect the same grid.

Example The following example toggles the Grid object's Visible property, making the grid visible and then not visible.

```
' Dimension the variables
Dim igxApp As Application
Dim igxGrid As Grid
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxGrid variable to the Grid object
Set igxGrid = igxApp.Grid
' Set the Visible Property to True to see the grid
MsgBox "Click OK to make the grid visible"
igxGrid.Visible = True
MsgBox "Click OK to make the grid not visible"
igxGrid.Visible = False
MsgBox "Click OK to continue"
```

See Also Grid object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Help Method

Syntax *Application*.**Help**([*HelpFileName* As String], [*HelpContext* As Variant])

Description The Help method displays a help file. The *HelpFileName* argument specifies the help file to open. If you omit the first argument, the help file shipped with iGrafx Professional is opened. The *HelpContext* argument is a context ID (an integer), which specifies a particular topic in the help file. If you omit the *HelpContext* argument, the Contents page of the help file is displayed. If the *HelpContext* argument's value cannot be resolved, the Help file's default topic is displayed.

Example The following example displays the help contents page of the iGrafx Professional help file.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Invoke the help file for iGrafx Professional
igxApp.Help
```

See Also Hint method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Hint Method

Syntax *Application*.**Hint**(*HintText* As String)

Description The Hint method allows you to specify a temporary message to be displayed in the status bar. The message stays in the hint line until the cursor moves over another item in iGrafx Professional that causes the hint line to change. This functionality is duplicated in the StatusBar.Text2 property. To set a more permanent message in the status bar, use the StatusBar.Text property. The *HintText* argument specifies the message to display.

Example The following example invokes the Hint Method to place a textual hint at the bottom left of the application window.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Invoke the Hint Method for the application
igxApp.Hint "Testing the Hint Method!"
MsgBox "See the hint line at the bottom of the application window."
```

See Also Help method

StatusBar object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

IsCommandAvailable Method

Syntax *Application*.**IsCommandAvailable** (*Command* As IxBuiltInCommand) As Boolean

Description The IsCommandAvailable method lets you determine whether a built-in command is enabled or available in the user interface. For example, the command Edit—Copy command (ixEditCopy) is not available when there is no selection in the active diagram, or there is no active diagram. Note that ExecuteCommand does not let you execute commands that are not available.

For the list of valid values for the IxBuiltInCommand constant, refer to the ExecuteCommand method.

Example The following example executes the Print Preview built-in command after first checking if the command is enabled or available in the user interface.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Check to see if Print Preview is available via the
' IsCommandAvailable Method.
If igxApp.IsCommandAvailable(ixFilePrintPreview) Then
    igxApp.ExecuteCommand ixFilePrintPreview
    MsgBox "Print preview"
Else
    MsgBox "Print preview command is not available"
End If
```

See Also ExecuteCommand property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Left Property

Syntax *Application*.**Left**

Data Type Long (read/write)

Description The Left property specifies the position of the left side of the iGrafx Professional application window in pixels. The number of pixels available, and therefore, valid settings for this property, depends on your screen resolution. For example, if you are running in standard VGA mode, your screen is 640 pixels wide and 480 pixels tall. Negative values are rounded to "1". Very large values could move the window off the edge of the display, making the window inaccessible.

Example The following example sets the application's left side using the Left property.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Sets the left side of the application in pixels
MsgBox "Click OK to set the Left side at 500 pixels."
igxApp.Left = 500
MsgBox "Click OK to set the Left side at 10 pixels."
igxApp.Left = 10
MsgBox "Click OK to continue"
```

See Also Top property

Width property

Window property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Maximize Method

Syntax *Application*.**Maximize**

Description The Maximize method puts the iGrafx Professional application window in Maximize mode, occupying the entire display. This method is equivalent to clicking the maximize button in the upper right corner of the iGrafx Professional window, or selecting the Maximize option from the Window menu.

Example The following example maximizes the application's window using the Maximize method.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Maximizes the application's window
MsgBox "Click OK to Maximize the window"
igxApp.Maximize
MsgBox "Click OK to Restore the original window size"
igxApp.Restore
MsgBox "Click OK to continue"
```

See Also

Minimize method

Restore method

WindowState property

Window object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Minimize Method

Syntax *Application*.**Minimize**

Description The Minimize method changes the iGrafx Professional window to an icon in the task bar. This method is equivalent to clicking the minimize button in the upper right corner of the iGrafx Professional window, or selecting the Minimize option from the Window menu.

Example The following example minimizes the application window using the Minimize method.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Maximizes the application's window
MsgBox "Click OK to Minimize the window"
igxApp.Minimize
MsgBox "Click OK to Restore the original window size"
igxApp.Restore
MsgBox "Click OK to continue"
```

See Also Maximize method

Restore method

WindowState property

Window object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Move Event

Syntax **Sub** *Application*_**Move**()

Description The Move event occurs when the position of the iGrafx Professional window changes. This can occur when either the Left prop[erty or the Top property is changed.

Example The following example illustrates when the move event is fired in response to code. Copy this entire code example into your diagram's code window, and run MoveEventTest.

```
' This declaration causes the new application object to listen to events
Public WithEvents igxApp As Application

Sub MoveEventTest()
    ' Create an application object
   Set igxApp = Application.Application
   ' Try various window resize methods
   igxApp.Top = 100    ' Move event occurs
   igxApp.Top = 100    ' Move event doesn't occur
                       ' since top is already 100
   igxApp.Left = 100   ' Move event occurs
   igxApp.Left = 100   ' Move event doesn't occur
                       ' since left is already 100
   igxApp.Height = 100 ' Move event doesn't occur
                       ' because height changes don't
                       ' affect the position
   igxApp.Width = 100  ' Move event doesn't occur
                       ' because width changes don't affect
                       ' the position
   igxApp.Maximize     ' Move event occurs because maximize
                       ' affects the position
   igxApp.Width = 500  ' Move event occurs because changing
                       ' the width of a maximized window
                       ' causes it to change out of the
                       ' maximize window position
End Sub

' Event handler for the Move event
Private Sub igxApp_Move()
    MsgBox "Move Event was fired"
End Sub
```

See Also Resize event

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

NewDocument Event

Syntax **Sub** *Application*_**NewDocument**(*Document* As Document)

Description The NewDocument event occurs whenever a new document is created. This event occurs once during the lifetime of a particular document; that is, it does not occur when a document is saved and then opened again. When the event fires, it passes a reference to the new document to the event handler subroutine.

You can put code in the NewDocument event to initialize a document in some way. If you are developing a custom document type, you can check the new document's document type in this event and apply any required initializations if the document type matches your custom document type. This event is also useful when the user needs to be notified of the creation of a new document.

Example The following example adds custom Property Lists to a new document. It uses the ExecuteCommand method to open a new document, which causes the NewDocument event to fire. The event handler code adds the Property Lists to the new document. To try this example code, copy all three sections as one block into your diagram's code window.

```
' Sets up igxApp as an Application object that will listen to events
Public WithEvents igxApp As Application

Sub NewDocEventTest()
    ' Create a new application object
   Set igxApp = Application.Application
   ' Open a new document.  This should fire the event.
   MsgBox "Click OK to open a new document"
   igxApp.ExecuteCommand (ixFileNew)
End Sub

' This is the event handler code
Private Sub igxApp_NewDocument(ByVal Document As Document)
    ' Add custom Property Lists to every new document
   With Document
       .PropertyLists.Add "MyCompany.PropertyList1.Tables"
       .PropertyLists.Add "MyCompany.PropertyList1.Queries"
       MsgBox "PropertyLists added to the new document"
   End With
End Sub
```

See Also OpenDocument event

Document object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

OpenDocument Event

Syntax **Sub** *Application*_**OpenDocument**(*Document* As Document)

Description The OpenDocument event occurs each time an existing document is opened. This event can be useful when the user needs to be notified when a document is opened.

Example The following example displays a message every time a document is opened. It uses the ExecuteCommand method to open a new document, which causes the OpenDocument event to fire. The event handler code then displays a message. This example requires that you have at lease one iGrafx Professional document saved to disk that you can open (you can use any of the files in the Exercise directory). To try this code, copy all three sections as one block into your diagram's code window.

```
' Sets up igxApp as an Application object that listens to events
Public WithEvents igxApp As Application

Sub OpenDocEventTest()
    ' Create a new application object
   Set igxApp = Application.Application
   ' Open a new document.  This should fire the event.
   MsgBox "Click OK to open a new document"
   igxApp.ExecuteCommand (ixFileOpen)
End Sub

' This is the event handler code
Private Sub igxApp_OpenDocument(ByVal Document As Document)
    MsgBox "A file called " & Document.Name & " has been opened."
End Sub
```

See Also NewDocument event

Document object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Output Method

Syntax *Application*.**Output**(*OutputText* As String, [*OutputPaneName* As String = “Output”])

Description The Output method writes a text string to the application’s Output window. The *OutputText* argument is the string to print. The optional *OutputPaneName* argument specifies the name of an output pane. If no output pane is specified, by default, the method writes the string to a pane named “Output”, which is created automatically on the first use of this method. If the name of the OutputPane you specify does not exist, it is created automatically.

For more control over the output window, you can use the OutputWindow, OutputPanes, and OutputPane objects.

Example The following example write a string of text to an output pane named “MyPane”. If the out pane does not exist, it is created. Note that the “Call” keyword is not necessary if you only specify the *OutputText* argument, or you explicitly name the arguments.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Output a string to the Output window. The method specifies a
' non-existent output pane
Call Output("This is a test of the Output window", "MyPane")
```

See Also OutputWindow object

OutputPane object

OutputPanes object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

OutputWindow Property

Syntax *Application*.**OutputWindow**

Data Type OutputWindow object (read-only, See Object Properties)

Description The OutputWindow property returns the OutputWindow object for the Application object. This object represents the iGrafx Professional Output window. You can use the Output window to write information you want to keep track of or display to a user.

Example The following example make the Output window visible and creates a new output pane named “My Pane #1”.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Shows the output window
igxApp.OutputWindow.Visible = True
' Add an output pane named "My Pane #1"
igxApp.OutputWindow.OutputPanes.Add "My Pane #1"
MsgBox "You will now find a new Output Window tab" _
    & Chr(13) & "at the bottom left of the application window."
```

See Also OutputWindow object

OutputPane object

OutputPanes object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

OutputWindowGoTo Event

Syntax **Sub** *Application*_**OutputWindowGoTo**(ByVal *Key* As Long, *Handled* As Boolean)

Description The OutputWindowGoTo event occurs when a user double clicks on a line of output in the output window. All strings that are written to the output window have an associated “Key” value. When a line of text is double-clicked, the Key value is passed to the *Key* parameter of this event.

The main purpose of this event is for setting up code that performs some actions in response to the user double-clicking an output window message you have placed there. You capture the key in a global variable, and then if the *Key* parameter matches one of the keys you want to respond to, the code is run.

The *Handled* parameter is used to indicate whether the event has been handled. Remember that there could be any number of extension projects or DLLs that are listening to this event. If the key is one your event code responds to, then as a courtesy, you should set the *Handled* parameter to True. You can also check the state of the *Handled* parameter to *potentially* determine whether someone else’s code has handled the event.

For information about how keys are generated, refer to the OutputPane.AddString method.

Example The following example illustrates the use of the OutputWindowGoTo event to put shortcut commands in an Output window. An empty output window is created, and two lines of text are displayed in the new output pane. The application object is listening to events, and when the user double clicks a line of text in the output window, the OutputWindowGoTo event is fired. To try this example code, copy all three sections as one block into a Diagram project, or into an Extension Project.

```
' Sets up igxApp as an Application object that listens to events
Public WithEvents igxApp As Application
' Dimension key number variables as global variables
Dim lOWKey1, lOWKey2 As Long

Sub EventTest()
   ' Create a new application object
   Set igxApp = Application.Application
    ' Delete any current output panes
   For Index = 1 To igxApp.OutputWindow.OutputPanes.Count
        igxApp.OutputWindow.OutputPanes.Item(Index).Delete
   Next Index
   igxApp.OutputWindow.OutputPanes.Add ("ShortCuts")
   igxApp.OutputWindow.Visible = True
   igxApp.OutputWindow.OutputPanes.Item(1).Activate
   ' Add text to the output window and store the key numbers
   lOWKey1 = igxApp.OutputWindow.OutputPanes.Item(1).AddString _
        ("Display the Fill Color gallery")
   lOWKey2 = igxApp.OutputWindow.OutputPanes.Item(1).AddString _
        ("Display the Font gallery")
   MsgBox "To test the output window item, return to the diagram," _
        & Chr(13) & "and double click the line in the output window."
End Sub

Private Sub igxApp_OutputWindowGoTo(ByVal Key As Long, Handled As Boolean)
   ' Display the Fill Color gallery
   If Key = lOWKey1 Then
        Application.Gallery.GalleryPanes.Item(1).Activate
        Handled = True
```

```
    End If
    ' Display the Font gallery
    If Key = lOWKey2 Then
         Application.Gallery.GalleryPanes.Item(2).Activate
         Handled = True
    End If
End Sub
```

See Also

OutputWindow object

OutputPane object

OutputPanes object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Path Property

Syntax *Application*.**Path**

Data Type String (read-only)

Description The Path property allows the programmer to retrieve the file system path name for the iGrafx Professional application, excluding the executable file name. The path does not include a final back slash. To get the path with the executable file name, use the FullName property.

Example The following example retrieves the application's path from the Path property, and displays the path in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
Dim strPath As String
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Retrieves the path of the application
strPath = igxApp.Path
' Displays the path of the Application
MsgBox "The application's path is " & strPath
```

See Also

Build property

Caption property

FullName property

Name property

Path property

Version property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

PercentGauge Property

Syntax *Application*.**PercentGauge**

Data Type PercentGauge object (read-only, See Object Properties)

Description The PercentGauge property returns the PercentGauge object. The object represents a percent gauge dialog. The programmer should use a percent gauge during any operation that takes longer than a few seconds to show the user the progress of an operation.

Example The following example launches the PercentGauge dialog and increments the gauge.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Make the PercentGauge dialog visible
igxApp.PercentGauge.Visible = True
' Place text on the percent gauge
igxApp.PercentGauge.Text = "Processing"
' Create a loop to increment the gauge based on the system Timer
While (igxApp.PercentGauge.Value < 100)
    If (Timer <> LastTimer) Then
        igxApp.PercentGauge.Value = igxApp.PercentGauge.Value + 1
        LastTimer = Timer
    End If
    ' DoEvents allows the system to process other events and
    ' prevents the percent gauge loop from hanging the system
    DoEvents
Wend
igxApp.PercentGauge.Visible = False
```

See Also PercentGauge object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

PopupWindows Property

Syntax *Application*.**PopupWindows**

Data Type PopupWindows collection object (read-only, See Object Properties)

Description The PopupWindows property returns the PopupWindows collection for the Application object. The PopupWindows collection contains all the current windows that are not MDI child windows of the iGrafx Professional application window. It allows the programmer to get individual PopupWindow objects and manipulate them.

Controlling the display of various PopupWindows is important when you are displaying your own dialogs and you want to make sure that none of the modeless windows are floating on top of your dialog.

Example The following example displays the current number of popup windows in the collection.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Display the number of pop up windows
MsgBox "The current number of Popup Windows is " & igxApp.PopupWindows.Count
```

See Also PopupWindow object

PopupWindows object

Window object

Windows object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Quit Event

Syntax **Sub** *Application*_**Quit**()

Description The Quit event occurs before the iGrafx Professional application shuts down. A programmer can make use of this event to perform any clean-up operations that are necessary before the application is closed.

Example The following example sets up an Application object that listens to events. The application is then shut down using the QuitApplication method. Before the application quits, the Quit event is fired.

The simplest way to implement application events is to put an event handler subroutine in the **ExtensionProject->ThisApplication** project code window in the Visual Basic editor. For more information, refer to the discussion of the ExtensionProject object.

```
' Sets up igxApp as an Application object that listens to events
Public WithEvents igxApp As Application
```

Put this code into the **ExtensionProject->ThisApplication** project code window, and then select **File->Save** Extension. Now when the application is shut down, your event code executes.

```
Private Sub igxApp_Quit()
    MsgBox "The Quit Event was fired. iGrafx Professional is signing off."
End Sub
```

See Also QuitApplication method

ExtensionProject object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

QuitApplication Method

Syntax *Application*.**QuitApplication**

Description The QuitApplication method closes iGrafx Professional. It does not prompt the user to save changes to open files; therefore, the programmer should be careful using this method, and if saving changes to open files is important, take appropriate steps prior to invoking this method.

Example The following example invokes the QuitApplication method to quit the application. Note that this example shuts down iGrafx Professional without saving. Do NOT run this example with any important iGrafx Professional documents open.

```
' Create a new application object
Set igxApp = Application.Application
' If the users answers OK, shut down iGrafx Professional
If MsgBox("This will shut down iGrafx Professional. All information" _
   & Chr(13) & "will be lost. Do you want to shut down " _
   "iGrafx Professional?", vbOKCancel) = vbOK Then
       ' Shut down iGrafx Professional immediately, no save
       igxApp.QuitApplication
End If
```

See Also Quit event

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

RecentFiles Property

Syntax *Application*.**RecentFiles**

Data Type RecentFiles collection object (read-only, See Object Properties)

Description The RecentFiles property returns the RecentFiles collection for the Application object. The object contains the names (as strings) of the most recently opened files. These filenames are also displayed in the iGrafx Professional File menu. You can use the RecentFiles collection to examine, add to, remove from, and reorder the list of recently opened files.

Example The following example displays all the filenames in the RecentFiles collection.

```
' Dimension the variables
Dim igxApp As Application
Dim igxRecentFiles As RecentFiles
Dim sNames As String
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxRecentFiles variable to the RecentFiles collection
Set igxRecentFiles = igxApp.RecentFiles
For Index = 1 To igxRecentFiles.Count
   sNames = sNames & igxRecentFiles.Item(Index) & Chr(13)
Next Index
' Display the number of items in the RecentFiles collection
MsgBox "Recently opened files:" + Chr(13) + sNames
```

See Also RecentFiles object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

RefreshUI Method

Syntax *Application*.**RefreshUI**

Description The RefreshUI method forces the user interface to refresh. It refreshes the toolbars, window tabs, and other user interface elements. Using this method is important after implementing code that changes or alters some aspect of the user interface display.

Example The following example retrieves the “Standard” command bar from the CommandBars collection, and determines the value of its Position property. Then the command bar’s position is changed with the Left and Top properties. Message boxes are used so that the results can be seen. Note that the Application.RefreshUI method is needed so that the UI refreshes while the code is running. Try running this example without using RefreshUI to see what happens.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim sPosition As String
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Determine the setting of the Position property
Select Case igxCmdBar.Position
    Case ixDockTop:
        sPosition = "Docked at Top"
    Case ixDockBottom:
        sPosition = "Docked at Bottom"
    Case ixDockLeft:
        sPosition = "Docked at Left"
    Case ixDockRight:
        sPosition = "Docked at Right"
    Case ixFloating:
        sPosition = "Floating--Not Docked"
End Select
MsgBox "View the position of the " & igxCmdBar.Caption _
    & " command bar." & Chr(13) & "Its Position is " _
    & sPosition
' Set the Left property to 50 pixels, and Top to 100 pixels
igxCmdBar.Left = 50
igxCmdBar.Top = 100
MsgBox "The position of the " & igxCmdBar.Caption _
    & " command bar has been moved 50 pixels to the right," _
    & Chr(13) & "and down 100 pixels." & Chr(13) _
    & "Notice that the Commandbar has disappeared."
' Refresh the UI to make the Command Bar repaint
Application.RefreshUI
MsgBox "The Command Bar has reappeared, but has not been moved" _
    & Chr(13) & "down. This is because there is only one Command" _
    & Chr(13) & "bar docked at the Top."
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

RegisterExtension Method

Syntax *Application*.**RegisterExtension**(*Extension* As IGrafxExtension, [*Context* As IxExtensionContext = ixExtensionContextApplication], [*ContextGUID* As String])

Description The RegisterExtension method provides an entry point to register extensions that participate in iGrafx Professional's extension architecture. For more information, see the iGrafx System Developer's Guide.

The *Extension* argument is a VBA class that you have written that implements an IGrafxExtension object.

The *Context* argument specifies an extension context. The extension architecture is built around the notion of contexts. The IxExtensionContext constant defines the valid values, which are listed in the following table.

Value	Name of Constant
1	ixExtensionContextApplication
2	ixExtensionContextCustomContext
3	ixExtensionContextAllDocuments
4	ixExtensionContextDefaultDocument
5	ixExtensionContextCustomDocument
6	ixExtensionContextAllViews
7	ixExtensionContextDefaultView
8	ixExtensionContextCustomView
9	ixExtensionContextDiagram
10	ixExtensionContextCustomDocComponent

The *ContextGUID* argument is used for certain contexts that require a GUID.

The extension architecture is beyond the scope of this help file. It is described in detail in the iGrafx Extensions Development Guide.

Example The following example show how to register an extension with iGrafx Professional. First create a new class called Class1. This class implements IGrafxExtension, which has two required methods: ContextBegin, and ContextEnd.

```
' Class1
' Need to implement IGrafxExtension in some class before
' registering an extension
Implements IGrafxExtension
' This method controls what happens when the extension enters the context
Private Sub IGrafxExtension_ContextBegin(ByVal Host As IXFlowExtensionHost)
   ' Stand in method
End Sub
' This method controls what happens when the extension leaves the context
Private Sub IGrafxExtension_ContextEnd()
   ' Stand in method
End Sub
```

Next put the following code into a project code window. This code registers an extension using the new class.

```
Sub main()
```

```
    ' Dimension the variables
    Dim igxApp As Application
    Dim igxExt As New Class1
    ' Set the ixapp App variable to the current Application object
    Set igxApp = Application.Application
    ' Register a new Extension
    igxApp.RegisterExtension igxExt, ixExtensionContextApplication
    MsgBox "Extension Registered"
End Sub
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

RegisterTimer Method

Syntax *Application*.**RegisterTimer**(*Callback* As Callback, *IntervalInSeconds* As Integer) As Long

Description The RegisterTimer method allows you to write a VBA class that implements a Callback, and then have iGrafx Professional repeatedly call the class at a given interval.

To use the RegisterTimer method, first create a VBA class that implements a Callback. In your class's Execute method, write the code that you want executed on a periodic basis; for example, you might want to poll a database every 30 seconds and update the diagram.

When you register the timer, you pass in an instance of your class. You also need to specify the *IntervalInSeconds* parameter, which sets how often your class is called by iGrafx Professional.

The RegisterTimer method returns a Long value. This value is known as a "cookie." To unregister your timer (and stop iGrafx Professional from calling you back again), you pass your cookie into the UnregisterTimer method.

Example The following example changes the color of the first diagram object every 2 seconds, using a Timer. To try this example, copy each block of code as indicated.

The following code implements a class with a callback. Within a Diagram project, create a new class called Class1, and copy this code into it.

```
' Class1
' Force explicit variable declarations (corrects scope)
Option Explicit
' Need to specify that this class uses callback
Implements Callback
' The Execute subroutine is what runs when the timer hits the callback class
Private Sub Callback_Execute()
    ' Change color of the diagram object at random
    Application.ActiveDiagram.DiagramObjects.Item(1).Shape _
        .FillColor = RGB(Rnd(1) * 255, Rnd(1) * 255, Rnd(1) * 255)
End Sub
```

The following code is the main program. Copy this block into the Diagram project code window and run it.

```
Public Sub TestTimer()
    ' Dimension the variables
    Dim igxTimer As Class1
    Dim igxShape1 As Shape
    ' Create an instance of our timer class
    Set igxTimer = New Class1
    ' Put a shape on the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440, 1440)
    ' Start the timer, and set the cookie value
    ' It is possible to install multiple timers
    ' Each one would have a different cookie value
    ' This timer fires every 2 seconds
    TimerCookie = Application.RegisterTimer(igxTimer, 2)
    MsgBox "Click OK to stop the timer."
    ' Stop the timer using the cookie value
    Application.UnregisterTimer TimerCookie
End Sub
```

See Also UnregisterTimer method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

RepaintAll Method

Topic Under Construction!!!

Syntax *Application*.**RepaintAll**

Description The RepaintAll method forces a repaint of all the application's windows.

Example The following example

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Resize Event

Syntax **Sub** *Application*_**Resize**()

Description The Resize event occurs when the iGrafx Professional application window is resized (the Width or Height properties are changed). A programmer can make use of this event to perform actions based on a user resizing the application window.

Example The following example adjusts the position of a Diagram object based on the size of the application window. The event handler constantly repositions the shape toward the right side of the window area as the application window is resized. To try this code, copy all three sections as one block into your diagram's code window, and execute the EventTest subroutine.

```
' Sets up igxApp as an Application object that listens to events
Public WithEvents igxApp As Application

Sub EventTest()
    ' Create a new application object
    Set igxApp = Application.Application
    MsgBox "Now return back to the diagram and resize the " _
        & "application window" & Chr(13) & "using the mouse. " _
        & "The Resize Event contains code which" _
        & Chr(13) & "updates the position of the first diagram " _
        & "shape to keep it toward" & Chr(13) _
        & "the right side of the window area as the window " _
        & "is resized."
End Sub

Private Sub igxApp_Resize()
    'Adjust the position of a diagram object based on the window size
    With igxApp.ActiveDiagram.DiagramObjects.Item(2)
        .CenterX = (igxApp.Window.Width - 20) * 8
        .Shape.Text = "CenterX is: " & .CenterX
    End With
End Sub
```

See Also Height property

Move event

Width property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Restore Method

Syntax *Application*.**Restore**

Description The Restore method restores the iGrafx Professional window to its previous state, either maximized or normal. This method is equivalent to clicking the restore button in the upper right corner of the iGrafx Professional window.

Example The following example illustrates the use of the Restore method.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object.
Set igxApp = Application.Application
' Maximizes the application's window.
MsgBox "Click OK to Maximize the window"
igxApp.Maximize
MsgBox "Click OK to Restore the original window size"
igxApp.Restore
MsgBox "Click OK to continue"
```

See Also Maximize method

Minimize method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Ruler Property

Syntax *Application*.**Ruler**

Data Type Ruler object (read-only, See Object Properties)

Description The Ruler property returns the Ruler object for the Application object. The object represents the ruler that is shown when the Rulers option is checked in the iGrafx Professional View menu. The Ruler object allows you to show, hide, and customize the ruler.

Example The following example demonstrates how to hide the ruler.

```
' Dimension the variables
Dim igxApp As Application
Dim igxRuler As Ruler
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxRuler variable to the Ruler object
Set igxRuler = igxApp.Ruler
' Make the ruler not visible
igxRuler.Visible = True
MsgBox "Click OK to hide the ruler"
igxRuler.Visible = False
MsgBox "Click OK to make the ruler visible"
igxRuler.Visible = True
MsgBox "Click OK to continue"
```

See Also Ruler object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

SecurityLevel Property

Syntax *Application*.**SecurityLevel**

Data Type IxSecurityLevel enumerated constant (read-only)

Description The SecurityLevel property returns the security level of the application. The SecurityLevel property directly coresponds to the security levels used in Microsoft Internet Explorer.

Any changes you make to the security level are only in effect until iGrafx Professional is closed. To be polite (and secure), you should always return the SecurityLevel to the value it was at before you changed it.

This property is provided for the benefit of macros that need to run without displaying any dialogs. By setting the security level to ixSecurityNone, security dialogs will not appear and obstruct the execution of the macro.

The IxSecurityLevel constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant	Description
1	ixSecurityNone	No warning before loading documents that contain macros.
2	ixSecurityMedium	Warn before loading documents that contain macros.
3	ixSecurityHigh	Don't allow macros to be loaded when opening a document.

Example The following example displays the current security level.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Display the current security level
MsgBox "The current security level is " & igxApp.SecurityLevel
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ShapeLibraries Property

Syntax *Application*.**ShapeLibraries**

Data Type ShapeLibraries collection object (read-only, See Object Properties)

Description The ShapeLibraries property returns the ShapeLibraries collection for the Application object. This object represents a collection of the shape palettes that are available to the application.

Each ShapeLibrary object represents a shape palette. These shapes are globally available and are stored by the iGrafx Share Media applet. ShapeLibrary objects are added to the application using the ShapeLibraries.Add method, and by using "Open Shape Palette (F9)" in the user interface.

Note: The ShapeLibraries object appears at the application level, but there is another ShapeLibraries object at the Document level. The following information is about the Document level ShapeLibraries object.

A Document object can contain one or more Diagram Types. Each DiagramType object has a ShapeLibrary object that represents a shape palette. Each ShapeLibrary associated with a specific DiagramType is stored internally in the document. Therefore, if a document is sent to another site that doesn't have the same globally available shapes, the shapes in the document are available to the recipient.

The reason that each DiagramType has its own ShapeLibrary is because you could have a situation where a document contains both organizational diagrams and process maps, and a different shape palette may be needed for each type of diagram. In other words, a ShapeLibrary provides a way to organize shapes according to the type of diagram that uses those shapes.

Example The following example adds two Shape Libraries at the application level, and displays the subject names of the Libraries.

```
' Dimension the variables
Dim igxApp As Application
Dim igxShapeLibraries As ShapeLibraries
Dim sNames As String
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxShapeLibraries variable to the ShapeLibraries collection
Set igxShapeLibraries = igxApp.ShapeLibraries
'Add two shape libraries
MsgBox "Click OK to add two Shape Libraries at the application level."
igxShapeLibraries.Add "People", "Historic"
igxShapeLibraries.Add "Office", "Computer1"
'Collect the subject names in a string
For Index = 1 To igxShapeLibraries.Count
   sNames = sNames + igxShapeLibraries.Item(Index).SubjectName + Chr(13)
Next Index
' Display the subject names
MsgBox "The current Shape Library subjects:" & Chr(13) & Chr(13) & sNames
```

See Also

ShapeLibrary object

ShapeLibraries object

ShapeLibraryItem object

DiagramType object

DiagramTypes object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

ShowFinished Property

Syntax *Application*.**ShowFinished**[= {**True | False**}]

Data Type Boolean (read/write)

Description The ShowFinished property specifies whether to display the Finished button in the user interface in order to end a particular program mode. The Finished button is a user interface feature that makes it easier for users to know when they are in a particular mode, such as the shape drawing mode or the connector line drawing mode. Modes such as these are active until the user decides to be “finished” with that mode. The Finished button is a floating button that the user clicks to exit the mode. Advanced users may prefer to have this user interface feature turned off.

The following is a picture of what the Finished button looks like. The caption and the button text are handled by iGrafx Professional, and cannot be changed by the developer.

Example The following example toggles the ShowFinished property. When a mode is entered that would normally show the "Finished" button, the button is not displayed.

```
' Dimension the variables
Dim igxApp As Application
' Set the ixappApp variable to the current Application object
Set igxApp = Application.Application
' Set the ShowFinished Property to False
MsgBox "Click OK to turn the Finished button on"
igxApp.ShowFinished = True
igxApp.ActiveMode = ixDrawRectangle
MsgBox "Click OK to continue."
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Startup Event

Syntax **Sub** *Application*_**Startup**()

Description The Startup event occurs when iGrafx Professional is starting up, just before the Welcome dialog is displayed. You can write code in the Startup event to initialize application level extensions, toolbars, and other application level changes.

Example The following example sets up the Startup event to display a message when the application starts. To try this code, copy it into the Extension Project called "**This Application**". Then from the Visual Basic file menu, choose **File->SaveExtensionProject**. Now when you start the application, the event fires.

```
Private Sub Application_Startup()
    MsgBox "Welcome to my custom setup of iGrafx Professional"
End Sub
```

See Also ExtensionProject object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

StatusBar Property

Syntax *Application*.**StatusBar**

Data Type StatusBar object (read-only, See Object Properties)

Description The StatusBar property returns the StatusBar object for the Application object. The object represents the status bar at the bottom of the iGrafx Professional window.

Example The following example demonstrates how to hide the status bar.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the StatusBar's Visible Property to False
MsgBox "Click OK to hide the status bar."
igxApp.StatusBar.Visible = False
MsgBox "Click OK to make the status bar visible."
igxApp.StatusBar.Visible = True
MsgBox "Click OK to continue"
```

See Also Hint method

StatusBar object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Templates Property

Syntax *Application*.**Templates**

Data Type Templates collection object (read-only, See Object Properties)

Description The Templates property returns the Templates collection for the Application object. This object contains all of the Template objects that are available to the application. By using the Templates collection and the Template object, the developer can create a new document from any of the available templates.

Example The following example creates a new document derived from the Cascade Template.

```
' Dimension the variables
Dim igxApp As Application
Dim igxTemplates As Templates
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxTemplates variable to the Templates collection
Set igxTemplates = igxApp.Templates
' Point the template path to the iGrid templates
igxTemplates.DefaultTemplatePath = _
   "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid"
' Create a new document from a the Cascade template: Item(3)
MsgBox "Click OK to create a new document from the first Template"
igxTemplates.Item(3).OpenAsDocument
MsgBox "Document created. Click OK to continue."
```

See Also Template object

Templates object

Document object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

TileHorizontal Method

Syntax *Application*.**TileHorizontal**

Description The TileHorizontal method tiles all of the currently open windows horizontally within the iGrafx Professional application window. The windows are arranged so that a portion of each one is visible. The method works with one or more windows, and provides the same functionality as selecting the Tile Horizontal option from the Window menu.

Example The following example invokes the TileHorizontal method of the Application object to arrange all open windows in a horizontal manner.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Create a new document in the application
igxApp.Documents.New
MsgBox "New document added. Click OK to tile the windows horizontal."
' Invokes the cascade method for arranging the open windows
igxApp.TileHorizontal
MsgBox "Windows have been tiled horizontal."
```

See Also ArrangeIcons method

Cascade method

TileVertical method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

TileVertical Method

Syntax *Application*.**TileVertical**

Description The TileVertical method tiles all of the currently open windows vertically within the iGrafx Professional application window. The windows are arranged so that a portion of each one is visible. The method works with one or more windows, and provides the same functionality as selecting the Tile Vertical option from the Window menu.

Example The following example invokes the TileVertical method of the Application object to arrange all open windows in a vertical manner.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Create a new document in the application
igxApp.Documents.New
MsgBox "New document added. Click OK to tile the windows vertical."
' Invokes the cascade method for arranging the open windows
igxApp.TileVertical
MsgBox "Windows have been tiled vertical."
```

See Also ArrangeIcons method

Cascade method

TileHorizontal method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

TrialVersion Property

Syntax *Application*.**TrialVersion**[= {**True | False**}]

Data Type Boolean (read-only)

Description The TrialVersion property specifies whether the application is running in a trial version mode. While in trial version mode, saving of documents is disabled.

Example The following example displays a message indicating a trial or a retail version of iGrafx Professional.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Create a new document in the application
If igxApp.TrialVersion Then
    MsgBox "This is a trial version."
Else
    MsgBox "This is iGrafx Professional version " & igxApp.Version _
        & ", and not a trial version"
End If
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

UnregisterTimer Method

Syntax *Application*.**UnregisterTimer**(*Cookie* As Long)

Description The UnregisterTimer method unregisters a timer that was previously registered with the application using the RegisterTimer method. The RegisterTimer method returns a long value that is known as a "Cookie". When you specify that cookie to the UnregisterTimer method, iGrafx Professional stops calling into the class you specified in the RegisterTimer method.

Example The following example changes the color of the first diagram object every 2 seconds, using a Timer. To try this example, copy each block of code as indicated.

The following code establishes a class with a callback. Within a Diagram project, create a new class called Class1, and copy this code into it.

```
' Class1
' Force explicit variable declarations (corrects scope)
Option Explicit
' Need to specifyt that this class uses callback
Implements Callback
' The Execute subroutine is what runs when the timer hits the callback class
Private Sub Callback_Execute()
    ' Change color of the diagram object at random
   Application.ActiveDiagram.DiagramObjects.Item(1).Shape _
       .FillColor = RGB(Rnd(1) * 255, Rnd(1) * 255, Rnd(1) * 255)
End Sub
```

The following code is the main program. Copy this block of code into the Diagram project code window, and run it.

```
Public Sub TestTimer()
    ' Dimension the variables
   Dim igxTimer As Class1
   Dim igxShape1 As Shape
   ' Create an instance of the timer class
   Set igxTimer = New Class1
   ' Put a shape on the diagram
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Start the timer, and set the cookie value
   ' It is possible to install multiple timers
   ' Each one would have a different cookie value
   ' This timer fires every 2 seconds
   TimerCookie = Application.RegisterTimer(igxTimer, 2)
   MsgBox "Click OK to stop the timer."
   ' Stop the timer using the cookie value
   Application.UnregisterTimer TimerCookie
End Sub
```

See Also RegisterTimer method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

UserCompany Property

Syntax *Application*.**UserCompany**

Data Type String (read-only)

Description The UserCompany property returns the company name that is displayed in the About Box of the application. The UserCompany is stored in the system registry, and typically is set during installation. This is the company name that is provided when iGrafx Professional is installed.

Example The following example retrieves the user's company name, and displays the name in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
' Set the ixappApp variable to the current Application object
Set igxApp = Application.Application
' Display the user's company name that appears in the About Box
MsgBox "The user's company name is " & igxApp.UserCompany
```

See Also UserName property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

UserEvent Event

Syntax **Private Sub** *Application*_**UserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The UserEvent event occurs when someone uses the FireUserEvent method to fire the event. You can use the UserEvent event and the FireUserEvent method together to add additional events to the ones that are already available for iGrafx Professional objects. An Application-level UserEvent event should reside in the “ThisApplication” module of an Extension Project.

The UserEvent (and FireUserEvent method) takes as parameters an *EventIdentifier* string and a *Parameter* that is a Variant. For an EventIdentifier string, you should pick a unique string that won't conflict with other developers who are also using this functionality. A good choice is your company name concatenated with your product name and the event name.

The *Parameter* is a variant, so one logical choice is to pass a VBA class as the parameter.

Example The following example defines a new user event called "ShowUsers". The *Parameter* that gets passed is a class, which has one property called Count. The event handler displays the passed parameter’s Count property.

The following code creates a simple class with one property. Create a new class within a diagram project called Class1, and copy this code into it.

```
' Class1
' It contains one property, read only
Public Property Get Count() As Long
   Count = 25
End Property
```

The following code is the main program. Copy this, and the UserEvent subroutine, into the diagram project.

```
' Run this subroutine to test the event
Public Sub Main()
   ' Fire the UserEvent
   Application.FireUserEvent "ShowUsers", New Class1
End Sub

' This event handler runs every time any FireUserEvent method
' is used in the system
Private Sub Application_UserEvent(ByVal EventIdentifier As String, ByVal
Parameter As Variant)
    ' Check if the Identifier string is the one we want
    If EventIdentifier = "ShowUsers" Then
       ' Redirect to Class1
       MsgBox "The number of users is " & Parameter.Count
    End If
End Sub
```

See Also FireUserEvent method

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

UserName Property

Syntax *Application*.**UserName**

Data Type String (read-only)

Description The UserName property returns the user's name that is displayed in the About Box of the application. The UserName is stored in the system registry, and typically is set during installation. This is the user name that is provided when iGrafx Professional is installed.

Example The following example retrieves the user's name, and displays the name in a Message Box.

```
' Dimension the variables
Dim igxApp As Application
' Set the ixappApp variable to the current Application object
Set igxApp = Application.Application
' Display the user's name that appears in the About Box
MsgBox "The user's company name is " & igxApp.UserCompany _
    & Chr(13) & "The user's name is " & igxApp.UserName & "."
```

See Also UserCompany property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

VBE Property

Syntax *Application*.**VBE**

Data Type Object (read-only)

Description The VBE property allows the programmer access to the Visual Basic Editor.

Example The following example displays the Add-ins from the Visual Basic Editor in a message box. The example shows the message box output only if there are Addins present in the system.

```
' Dimension the variables
Dim igxAddin As Object
' Loop through each addin in the Visual Basic Editor's Add-in collection
For Each igxAddin In Application.VBE.AddIns
    MsgBox igxAddin.Description, , "VBE Addin(s):"
Next igxAddin
```

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Version Property

Syntax *Application*.**Version**

Data Type String (read-only)

Description The Version property returns the version number of the iGrafx Professional application. For finer version information, use the Build property.

Example The following example retrieves the user's name, and displays the name in a message box.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Display the version number that appears in the About Box
MsgBox "The version number is " & igxApp.Version
```

See Also

Build property

Caption property

FullName property

Name property

Path property

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Window Property

Syntax *Application*.**Window**

Data Type Window object (read-only, See Object Properties)

Description The Window property returns the Window object that represents the main iGrafx Professional application window.

Example The following example retrieves the caption of the main iGrafx Professional window and displays it in a message box.

```
' Dimension the variables
Dim igxApp As Application
Dim igxWindow As Window
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxWindow variable to the Window object
Set igxWindow = igxApp.Window
' Display the caption of the main iGrafx Professional window
MsgBox "The caption of the main iGrafx Professional window is " &
igxWindow.Caption
```

See Also

Height property

Left property

Top property

Width property

WindowState property

Maximize method

Minimize method

Restore method

Window object

Windows object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Windows Property

Syntax *Application*.**Windows**

Data Type Windows collection object (read-only, See Object Properties)

Description The Windows property returns the Windows collection for the Application object. The Windows collection represents the currently open document, diagram, or component windows. These windows are MDI child windows of the main iGrafx Professional window. Other non-MDI child windows (popup windows) are found in the PopupWindows collection.

Example The following example arranges the child windows of the main iGrafx Professional window in a cascade manner.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Create a new document in the application
igxApp.Documents.New
MsgBox "New document added. Click OK to Cascade the windows."
' Invokes the cascade method for arranging the open windows
igxApp.Cascade
MsgBox "All windows have been Cascaded."
```

See Also

PopupWindows object

Window object

Windows object

DiagramView object

View object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

WindowState Property

Syntax *Application*.**WindowState**

Data Type IxWindowState enumerated constant (read/write)

Description The WindowState property controls the current state of the iGrafx Professional main window. Any window in the application is always in one of the following states: Normal, Minimized, or Maximized. To control the state of a specific window, use the WindowState property of that Window object.

The IxWindowState constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant
0	ixWindowNormal
1	ixWindowMaximized
2	ixWindowMinimized

Notes

Expression	Equivalent To
Application.WindowState = ixWindowNormal	Application.Restore
Application.WindowState = ixWindowMaximized	Application.Maximize
Application.WindowState = ixWindowMinimized	Application.Minimize

Example The following example minimizes the application's window by setting the WindowState property to a value of ixWindowMinimized.

```
' Dimension the variables
Dim igxApp As Application
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Minimizes the application's window
MsgBox "Click OK to minimize the application's window"
igxApp.WindowState = ixWindowMinimized
```

See Also

Maximize method

Minimize method

Restore method

Window object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

Workspace Property

Syntax *Application*.**Workspace**

Data Type Workspace object (read-only, See Object Properties)

Description The Workspace property returns the Workspace object for the Application object. The Workspace object provides options for loading and saving workspaces to disk. A workspace stores information about currently open documents and views. It stores the state and position of all open windows. This state is restored when you load the workspace later. If the application contains any documents that have not yet been saved to disk, the user is prompted for the names of files to save.

Example The following example saves the current workspace.

```
' Dimension the variables
Dim igxApp As Application
Dim igxWorkSpace As Workspace
' Set the igxApp variable to the current Application object
Set igxApp = Application.Application
' Set the igxWorkSpace variable to the WorkSpace object
Set igxWorkSpace = igxApp.Workspace
' Saves the current workspace
MsgBox "Click OK to save the current workspace"
igxWorkSpace.Save "TestSpace"
MsgBox "Workspace saved as TestSpace"
```

See Also Workspace object

{button Application object,JI(`igrafxrf.HLP',`Application_Object')}

AnyControls Object

The AnyControls object allows a programmer access to the “Any” controls:

- AnyDocument
- AnyDiagram
- AnyShape
- AnyShapeExtender
- AnyConnector
- AnyConnectorExtender
- AnyDepartment
- AnyDepartmentExtender
- AnyObject

These “AnyControls” are used only in the context of an event.

The AnyControls object allows you to listen to events that originate from all the objects in a Diagram or Document. For example, if you wanted to know each time a new Diagram was created, you could listen to the AnyControls.AnyDiagram object’s events. A new event fires for the AnyDiagram object each time a new Diagram is created.

As an example, consider the task of keeping a master list of the names of all the shapes in a single diagram. The easiest way to do this is to listen to the Create, Delete, and Rename events for the AnyObject control at the Diagram level. To change this and keep a master list of the names of all the shapes in every diagrams in a document, you just change the code to listen to the AnyObject at the Document level rather than the Diagram level.

The key thing to remember about AnyControls is that they only make sense in the context of an event. The AnyControls are dynamically bound to whatever object is currently firing an event. For example, you want to perform some action (for instance, display the shape’s text) any time that any shape in a diagram is double-clicked. You would write something like the following:

```
Public Sub AnyShape_BeforeDoubleClick(…)
   MsgBox AnyShape.Text
End Sub
```

Assume there are three shapes in the current document. When Shape 1 is double-clicked, the AnyShape object is bound to Shape 1, so MsgBox AnyShape.Text displays Shape 1’s Text property. When Shape 2 is double-clicked, the AnyShape object is bound to Shape 2, and MsgBox AnyShape.Text displays Shape 2’s Text property, and so forth.

After an event is over, the AnyShape object is bound to Nothing. Trying to access the properties and methods of an AnyControl outside an event results in an error. For example, if you tried to write the code MsgBox AnyShape.Text outside of an “AnyShape” event, you get an error. Also, if you were to hold on to an AnyShape object after the AnyShape event, perhaps by copying it to a global variable, that object would be useless. If you need to hold onto the shape that is bound to the AnyShape object during an event, you can use one of the “permanent” properties, which gives you a DiagramObject that remains valid after the event (the DiagramObject that the AnyShape was bound to during the event).

Trying to look at an AnyControls object in the Properties window does not show any results when an event involving that AnyControls object is not firing. However, if you set a breakpoint during an event (for example you set a breakpoint on MsgBox AnyShape.Text in the above example), then during that event, you could inspect the AnyShape (or more correctly, the shape that fired the event) using the Properties window.

Additional Notes

- The VBA developer has access to all of the AnyControls properties except AnyDiagram off of a Document or Diagram project item. For a developer working outside of VBA, for example in VB or C++, the AnyControls object is more useful.
- A VBA developer can use the AnyDiagram object and the WithEvents keyword to listen to all the Diagram events for a given document.

- A Document object and a Diagram object have an AnyObject object. The VBAProject items, Diagram and Document, expose the AnyControls object directly as controls off of the project item. This is the preferred way of using the AnyControls object for a VBA programmer.
- The AnyDiagram object has no meaning (returns an error) for when written as Diagram.AnyControls.AnyDiagram. It only works if you write Document.AnyControls.AnyDiagram.

Properties, Methods, and Events

All of the Properties, methods, and events for the AnyControls object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
AnyConnector		
AnyConnectorExtender		
AnyDepartment		
AnyDepartmentExtender		
AnyDiagram		
AnyDocument		
AnyObject		
AnyShape		
AnyShapeExtender		
Application		
Parent		

AnyConnector Property

Syntax *AnyControls*.**AnyConnector**

Data Type Connector object (read-only, See Object Properties)

Description The AnyConnector property returns the AnyConnector control from the specified object (where the specified object implicitly sets the scope of events which the AnyConnector control hears). This is an event-time only control. You should only use it when establishing an event sink. You should only refer to an AnyControl during an event.

Example The following example sets up a connector line object that listens to events. In the EventTest() subroutine, the connector line object is set to the document's AnyConnector object, which causes the connector line object to hear events coming from ANY connector line. The last routine is an event handler routine for the BeforeClick event. This event is fired whenever ANY connector line is clicked. The BeforeClick event handler displays a message in an output window every time a connector line is clicked. To try this example, copy all this code as one block into the VBA code window. Run the EventTest subroutine, and the event becomes active.

```
' Dimension an AnyConnector Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of objects. This declaration is at the module level
' (not inside a Sub)
Public WithEvents MyAnyConnector As ConnectorLine

' The main program. Run this Sub to establish the event
Public Sub EventTest()
   ' Create the Object
   ' Event monitoring was already enabled when the object was declared
   Set MyAnyConnector = Application.ActiveDiagram.AnyControls.AnyConnector
   ' Let's confirm the setup with a message
   MsgBox "The event is now active. Return to the diagram and try it."
   ' Open an output window for our demonstration
   Application.OutputWindow.OutputPanes.Add ("Messages")
   Application.OutputWindow.OutputPanes.Item(1).Activate
   Application.OutputWindow.Visible = True
   Application.OutputWindow.OutputPanes.Item(1).Activate
End Sub

Private Sub AnyConnector_BeforeClick(ByVal X As Double, ByVal Y As Double,
Cancel As Boolean)
   ' Whenever a connector line is clicked, inform the user via the
   ' output window
    Application.OutputWindow.OutputPanes.Item(1).AddString _
       "You clicked a connector line"
End Sub
```

See Also ConnectorLine object

iGrafx API Object Hierarchy

{button AnyControls object,JI(`igrafxrf.HLP',`AnyControls_Object')}

AnyConnectorExtender Property

Syntax *AnyControls*.**AnyConnectorExtender**

Data Type DiagramObject object (read-only, See Object Properties)

Description The AnyConnectorExtender property returns an AnyConnectorExtender object from the AnyControls object. The AnyConnectorExtender is a special DiagramObject that listens to events coming from ConnectorLine objects, but hears the DiagramObject-level events only, not the ConnectorLine-level events. It should only be used in the context of events.

The main purpose of the AnyConnectorExtender object is to access DiagramObject events from outside the iGrafx Professional API, such as from Visual Basic Professional or C++. AnyConnectorExtender compliments AnyConnector outside the iGrafx Professional API because the AnyConnector object (outside iGrafx) provides ConnectorLine-level events only. The AnyConnectorExtender completes the model by providing the DiagramObject-level events. The following tables describe the events these objects hear both inside and outside iGrafx Professional.

VBA Inside iGrafx

Object	Events it hears
AnyConnector	Connector DiagramObject events and Shape events
AnyConnectorExtender	Connector DiagramObject events

Other applications outside iGrafx:

Object	Events it hears
AnyConnector	Connector events
AnyConnectoreExtender	Connector DiagramObject events

Example The following example sets up an AnyConnectorExtender object that listens to events. In the EventTest() subroutine, the connector line object is set to the document's AnyConnectorExtender object, which causes the connector line object to hear events coming from ANY connector line DiagramObject. The last routine is an event handler routine for the BeforeClick event. This event is fired whenever any connector line DiagramObject is clicked. The BeforeClick event handler displays a message in an output window every time a connector line is clicked. To try this example, copy all this code as one block into the VBA code window. Run the EventTest subroutine, and the event becomes active.

```
' Dimension an AnyConnectorExtender Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of objects. This declaration is at the module level
' (not inside a Sub)
Public WithEvents MyAnyConnectorExtender As DiagramObject

' The main program. Run this Sub to establish the event
Public Sub EventTest()
   ' Create the Object
   ' Event monitoring was already enabled when the object was declared
   Set MyAnyConnectorExtender = _
       Application.ActiveDiagram.AnyControls.AnyConnectorExtender
   ' Confirm the setup with a message
   MsgBox "The event is now active. Create a Connector Line and try it."
   ' Open an output window for our demonstration
```

```
   Application.OutputWindow.OutputPanes.Add ("Messages")
   Application.OutputWindow.OutputPanes.Item(1).Activate
   Application.OutputWindow.Visible = True
   Application.OutputWindow.OutputPanes.Item(1).Activate
End Sub

Private Sub MyAnyConnectorExtender_BeforeClick(ByVal X As Double, ByVal Y As
Double, Cancel As Boolean)
   ' Whenever a connector line is clicked, inform the user via the
   ' output window
   Application.OutputWindow.OutputPanes.Item(1).AddString _
       "You clicked a connector line"
End Sub
```

See Also

DiagramObject object

iGrafx API Object Hierarchy

{button AnyControls object,JI(`igrafxrf.HLP',`AnyControls_Object')}

AnyDepartment Property

Syntax *AnyControls*.**AnyDepartment**

Data Type Department object (read-only, See Object Properties)

Description The AnyDepartment property returns the AnyDepartment control from the specified object (where the specified object implicitly sets the scope of events that the AnyDepartment control hears). This is an event-time only control. You should only use it when establishing an event sink. You should only refer to an AnyControl during an event.

Example The following example creates a Department object that listens to events. It gets set to the document's AnyDepartment object, so it hears events from any department in any diagram in the document. The last routine is an event handler for the Rename event. If a department is renamed, this code is executed.

```
' Dimension an AnyConnector Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of objects. This declaration is at the module level
' (not inside a Sub).
Public WithEvents MyAnyDepartment As Department

' The main program. Run this Sub to establish the event.
Public Sub EventTest()
    ' Create the Object
    ' Event monitoring was already enabled when the object was declared
    Set MyAnyDepartment = Application.ActiveDocument.AnyControls.AnyDepartment
    ' Confirm the setup with a message
    MsgBox "The event is now active. Return to the diagram " _
        & "and rename a department."
End Sub

Private Sub AnyDepartment_Rename(ByVal OldName As String)
    ' This code executes if a department is renamed
    MsgBox "A department was renamed"
End Sub
```

See Also Department object

iGrafx API Object Hierarchy

{button AnyControls object,JI(`igrafxrf.HLP',`AnyControls_Object')}

AnyDepartmentExtender Property

Syntax *AnyControls*.**AnyDepartmentExtender**

Data Type DiagramObject object (read-only, See Object Properties)

Description The AnyDepartmentExtender property returns an AnyDepartmentExtender object from the AnyControls object. The AnyDepartmentExtender is a special DiagramObject that listens to events coming from Departments, but hears the DiagramObject-level events only, not the Department-level events. It should only be used in the context of events.

The main purpose of the AnyDepartmentExtender object is to access DiagramObject events from outside the iGrafx Professional API, such as from Visual Basic Professional or C++. AnyDepartmentExtender compliments AnyDepartment outside the iGrafx Professional API because the AnyDepartment object (outside iGrafx) provides Department-level events only. The AnyDepartmentExtender completes the model by providing the DiagramObject-level events. The following tables describe the events these objects hear both inside and outside iGrafx Professional.

VBA Inside iGrafx

Object	Events it hears
AnyDepartment	Department DiagramObject events and Shape events
AnyDepartmentExtender	Department DiagramObject events

Other applications outside iGrafx:

Object	Events it hears
AnyDepartment	Department events
AnyDepartmentExtender	Department DiagramObject events

Example The following example creates an AnyDepartmentExtender object that hears events coming from any DiagramObject that is a Department. The AfterSize event is set up to display the new size of a Department after resizing.

```
' Dimension an AnyDepartmentExtender Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of objects. This declaration is at the module level
' (not inside a Sub).
Public WithEvents MyAnyDepartmentExtender As DiagramObject

' The main program. Run this Sub to establish the event.
Public Sub EventTest()
    ' Create the Object
    ' Event monitoring was already enabled when the object was declared
    Set MyAnyDepartmentExtender = _
       Application.ActiveDocument.AnyControls.AnyDepartmentExtender
    ' Confirm the setup with a message
    MsgBox "The event is now active. Return to the diagram " _
        & "and rename a department."
End Sub

' This event fires if a Department DiagramObject is resized
Private Sub MyAnyDepartmentExtender_AfterSize _
(ByVal Width As Double, ByVal Height As Double)
```

```
    MsgBox "The new size of the Department is: " & _
       Round(Width / 1440, 1) & " in. x " & _
       Round(Height / 1440, 1) & " in."
End Sub
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button AnyControls object,JI(`igrafxrf.HLP',`AnyControls_Object')}

AnyDiagram Property

Syntax *AnyControls*.**AnyDiagram**

Data Type Diagram object (read-only, See Object Properties)

Description The AnyDiagram property returns the AnyDiagram control from the specified object (where the specified object implicitly sets the scope of events which the AnyDiagram control hears). This is an event-time only control. You should only use it when establishing an event sink. You should only refer to an AnyControl during an event.

Example The following example sets up an Event Sink with the AnyDiagram object. The example implements the Activate Event, which is triggered every time a different diagram is selected, (brought to the front/gains the focus), such as when the Diagram Activate method is used, or when the user clicks on a different diagram with the mouse.

```
' Dimension a Diagram Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of objects. This declaration is at the module level
' (not inside a Sub)
Public WithEvents MyAnyDiagram As Diagram

' The main program. Run this Sub to establish the event
Public Sub EventTest()
   ' Create the Object. Event monitoring was already enabled
   ' when the object was declared
   Set MyAnyDiagram = Application.ActiveDocument.AnyControls.AnyDiagram
   ' Confirm the setup with a message
   MsgBox "The event is now active. Return to the diagram " _
       & "and try selecting either diagram."
   ' Add a diagram so we have at least two for triggering the event
   Application.ActiveDocument.Diagrams.Add ("New Diagram")
End Sub

Private Sub AnyDiagram_Activate()
   MsgBox "A new diagram has been activated."
End Sub
```

See Also Diagram object

iGrafx API Object Hierarchy

{button AnyControls object,JI(`igrafxrf.HLP',`AnyControls_Object')}

AnyDocument Property

Syntax *AnyControls*.**AnyDocument**

Data Type Document object (read-only, See Object Properties)

Description The AnyDocument property returns a special Document object that listens to events coming from any open Document. The AnyDocument object is only valid during events. It does not return a valid Document object outside of event subroutines. To grab the Document object that fired the event, use the PermanentDocument property within the event subroutine.

The AnyDocument object is valid only within the “ThisApplication” module inside of an Extension Project.

Example The following example sets up the AnyDocument Activate event. Every time a document is activated, the user is told what document they are now working with.

```
Private Sub Main()
    ' Dimension document variables
    Dim igxDocument1 As Document
    Dim igxDocument2 As Document
    ' Get the active document
    Set igxDocument1 = ActiveDocument
    ' Add one more document
    Set igxDocument1 = ThisApplication.Documents.New
    ' Activate each document
    igxDocument1.ActivateDocument
    igxDocument2.ActivateDocument
    ' Inform the user the event is ready
    MsgBox "The event is ready. Try activating each document."
End Sub

' This event fires whenever a document activates
Private Sub AnyDocument_Activate()
     MsgBox "You are now working with " & ActiveDocument.Name
End Sub
```

See Also Document object

iGrafx API Object Hierarchy

{button AnyControls object,JI(`igrafxrf.HLP',`AnyControls_Object')}

AnyObject Property

Syntax *AnyControls*.**AnyObject**

Data Type DiagramObject object (read-only, See Object Properties)

Description The AnyObject property returns the AnyObject control from the specified object (where the specified object implicitly sets the scope of events which the AnyObject control hears). This is an event-time only control. You should only use it when establishing an event sink. You should only refer to an AnyControl during an event.

Example The following example sets up an Event Sink with the AnyObject object. The example has implemented the BeforeDelete event for AnyObject which fires before any diagram object is deleted. Here the event is used to add an extra messure of safety by displaying a message box asking the user to confirm the delete.

```
' Dimension an DiagramObject Object that hears events.
' The "WithEvents" keyword switches on the event listening feature
' of objects. This declaration is at the module level
' (not inside a Sub)
Public WithEvents MyAnyObject As DiagramObject

' The main program. Run this Sub to establish the event
Public Sub EventTest()
    Dim igxOutputPane As OutputPane
   ' Create the Object. Event monitoring was already enabled
   ' when the object was declared
   Set MyAnyDiagramObject = Application.ActiveDocument.AnyControls.AnyObject
   ' Confirm the setup with a message
   MsgBox "The event is now active. Return to a " _
       & "diagram and try deleting objects."
End Sub

Private Sub AnyObject_BeforeDelete(Cancel As Boolean)
    ' Confirm deletion of an object
    ' Set Cancel = True to abort the deletion
   If MsgBox("Sure you want to delete this object?", _
       vbExclamation + vbOKCancel, "BeforeDelete Event") _
       = vbCancel Then
           Cancel = True
   Else
       ' Set Cancel = False to allow the deletion
       Cancel = False
   End If
End Sub
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button AnyControls object,JI(`igrafxrf.HLP',`AnyControls_Object')}

AnyShape Property

Syntax *AnyControls*.**AnyShape**

Data Type Shape object (read-only, See Object Properties)

Description The AnyShape property returns the AnyShape control from the specified object (where the specified object implicitly sets the scope of events which the AnyShape control hears). This is an event-time only control. You should only use it when establishing an event sink. You should only refer to an AnyControl during an event.

Example The following example sets up an Event Sink with the AnyShape object. The example has implemented the BeforeDelete event for AnyShape which will fire before any shape is deleted. Here the event is used to add an extra messure of safety by displaying a message box asking the user to confirm the delete. To try is example code, copy all three sections as one block onto a code window, and run the EventTest( ) subroutine.

```
' Dimension an Shape Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of objects. This declaration is at the module level
' (not inside a Sub)
Public WithEvents MyAnyShape As Shape

' The main program. Run this Sub to establish the event
Public Sub EventTest()
   ' Create the Object. Event monitoring was already enabled
   ' when the object was declared
   Set MyAnyShape = Application.ActiveDocument.AnyControls.AnyShape
   ' Let's confirm the setup with a message
   MsgBox "The event is now active. Return to a diagram and " _
       & "try deleting shapes."
End Sub

Private Sub AnyShape_BeforeDelete(Cancel As Boolean)
   ' Confirm deletion of a shape
   ' Set Cancel = True to abort the deletion
   If MsgBox("Sure you want to delete this shape?", _
       vbExclamation + vbOKCancel, "BeforeDelete Event") _
       = vbCancel Then
           Cancel = True
   Else
       ' Set Cancel = False to allow the deletion
       Cancel = False
   End If
End Sub
```

See Also Shape object

iGrafx API Object Hierarchy

{button AnyControls object,JI(`igrafxrf.HLP',`AnyControls_Object')}

AnyShapeExtender Property

Syntax *AnyControls*.**AnyShapeExtender**

Data Type DiagramObject object (read-only, See Object Properties)

Description The AnyShapeExtender property returns an AnyShapeExtender object from the AnyControls object. The AnyShapeExtender is a special DiagramObject that listens to events coming from Shapes, but hears the DiagramObject-level events only, not the Shape-level events. It should only be used in the context of events.

The main purpose of the AnyShapeExtender object is to access DiagramObject events that are relevant to shapes from outside the iGrafx Professional API, such as from Visual Basic Professional or C++. The AnyShapeExtender object is designed to compliment the AnyShape object outside of the iGrafx Professional application; that is, when using the iGrafx Professional Type Library. From non-VBA programming projects, the AnyShapeExtender object provides the DiagramObject-level events, and the AnyShape object provides Shape-level events. The following tables describe the events these objects hear inside and outside iGrafx Professional.

VBA Inside iGrafx

Object	Events it hears
AnyShape	Shape DiagramObject events and Shape events
AnyShapeExtender	Shape DiagramObject events

Other applications outside iGrafx:

Object	Events it hears
AnyShape	Shape events
AnyShapeExtender	Shape DiagramObject events

Example The following example sets up an event with the AnyShapeExtender object. The example has implemented the BeforeDelete event for AnyShapeExtender, which fires before any shape is deleted. Here the event is used to add an extra measure of safety by displaying a message box asking the user to confirm the delete. To try is example code, copy all three sections as one block into a Diagram project, and run the EventTest( ) subroutine.

```
' Dimension an Shape Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of objects. This declaration is at the module level
' (not inside a Sub)
Public WithEvents MyAnyShapeExtender As DiagramObject

' The main program. Run this Sub to establish the event
Public Sub EventTest()
   ' Create the Object. Event monitoring was already enabled
   ' when the object was declared
   Set MyAnyShapeExtender = _
       Application.ActiveDocument.AnyControls.AnyShapeExtender
   ' Confirm the setup with a message
    MsgBox "The event is now active. Return to a diagram and " _
        & "try deleting shapes."
End Sub

Private Sub MyAnyShapeExtender_BeforeDelete(Cancel As Boolean)
```

```
    ' Confirm deletion of a shape
    ' Set Cancel = True to abort the deletion
    If MsgBox("Sure you want to delete this shape?", _
    vbExclamation + vbOKCancel, "BeforeDelete Event") _
    = vbCancel Then
        Cancel = True
    Else
        ' Set Cancel = False to allow the deletion
        Cancel = False
    End If
End Sub
```

See Also DiagramObject object

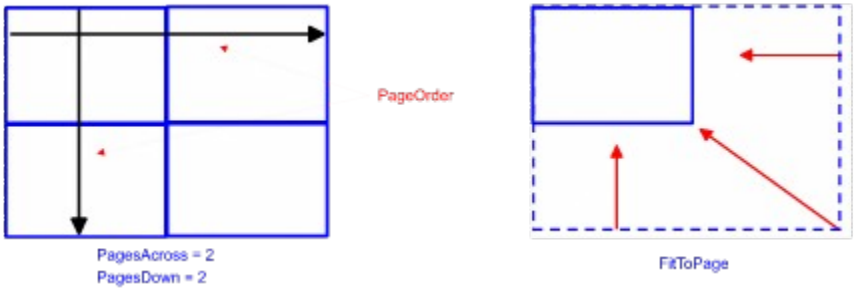
iGrafx API Object Hierarchy

{button AnyControls object,JI(`igrafxrf.HLP',`AnyControls_Object')}

PageLayout Object

The PageLayout object specifies the characteristics of a “page” in an iGrafx Professional diagram. This object provides the same functionality as the File – Page Setup option from the user interface (except for the Header/Footer tab). With this object, the developer can control and manipulate how iGrafx Professional handles pages.

The PageLayout object is uniquely associated with the Diagram object, and is part of what describes the characteristics of a diagram. Its purpose is solely for describing how a diagram is printed, or represented on a page. A diagram does not have an arbitrary size; it can be as large or small as it needs to be. Physical pages do have arbitrary sizes. The PageLayout object allows you to define how a diagram is mapped onto physical pages. Consider the following illustrations.

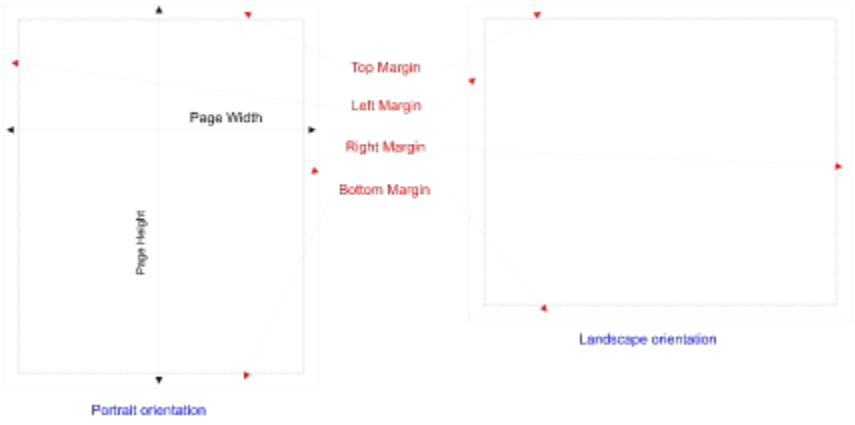


In the diagram on the left, the shaded block is the diagram, and the smaller rectangles outlined in blue represent pages. In this case, the diagram is mapped onto four 8.5” by 11.0” pages in landscape orientation. Three properties are also represented:

- The page order, which is either across first, or down first
- The number of pages to use horizontally across the diagram
- The number of pages to use vertically down the diagram

In the diagram on the right, a single page has been defined for a diagram that is larger than the page size. In this case, the FitToPage property can be used to force the diagram to be shrunk so it fits on the single page. If you did not use the FitToPage property, then only the upper left corner of the diagram would be printed on the page.

In the following diagram, the other properties that control page layout are depicted.



For information about pages and page layout, refer to the iGrafx Professional User’s Guide.

Properties, Methods, and Events

All of the properties, methods, and events for the PageLayout object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events

Application
BottomMargin
CenterFooter
CenterHeader
FitToPagesAcross
FitToPagesDown
FooterHeight
HeaderHeight
LeftFooter
LeftHeader
LeftMargin
Orientation
OverlapAmount
PageCount
PageHeight
PageOrder
PageTitleMode
PageWidth
PaperSize
Parent
PrintFrames
PrintNotes
RightFooter
RightHeader
RightMargin
ScalingMode
TopMargin
Zoom

Related Topics

Page object
iGrafx API Object Hierarchy

BottomMargin Property

Syntax *PageLayout*.**BottomMargin**

Data Type Long (read/write)

Description The BottomMargin property specifies the size of the bottom margin for all pages that comprise a diagram. This property provides the same functionality as the Page Setup option—Margins tab—Bottom Margin control through the user interface. The value of this property is specified in twips (1440 twips = 1.0 inch).

Example The following example shows how to set the margin properties of the PageLayout object for the ActiveDiagram by using a With statement. It sets the bottom and top margins to ½ inch, and the right and left margins to ¼ inch.

```
With Application.ActiveDiagram.PageLayout
    ' Set the top and bottom margins to 1/2 inch
    .BottomMargin = 1440 / 2
    .TopMargin = 1440 / 2
    ' Set the left and right margins to 1/4 inch
    .RightMargin = 1440 / 4
    .LeftMargin = 1440 / 4
End With
```

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

CenterFooter Property

Syntax *PageLayout*.**CenterFooter**

Data Type HeaderFooter object (read-only, See Object Properties)

Description The CenterFooter property returns a HeaderFooter object that is used to control the properties and format for the center region of the page footer. The specified properties and formatting are applied to every page of the diagram. The HeaderFooter object controls the same behavior that can be found in the Header/Footer tab of the Page Setup dialog.

Example The following example gets the HeaderFooter object for the active diagram and sets the Footer text.

```
' Dimension the variables
Dim igxHeaderFooter As HeaderFooter
' Retrieve the HeaderFooter object using the CenterFooter property
Set igxHeaderFooter = ActiveDiagram.PageLayout.CenterFooter
' Set the text for the Footer
igxHeaderFooter.Text = "Center Footer"
```

See Also HeaderFooter object

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

CenterHeader Property

Syntax *PageLayout*.**CenterHeader**

Data Type HeaderFooter object (read-only, See Object Properties)

Description The CenterHeader property returns a HeaderFooter object that is used to control the properties and format for the center region of the page header. The specified properties and formatting are applied to every page of the diagram. The HeaderFooter object controls the same behavior that can be found in the Header/Footer tab of the Page Setup dialog.

Example The following example gets the HeaderFooter object for the active diagram and sets the Header text.

```
' Dimension the variables
Dim igxHeaderFooter As HeaderFooter
' Retrieve the HeaderFooter object using the CenterFooter property
Set igxHeaderFooter = ActiveDiagram.PageLayout.CenterHeader
' Set the text for the Header
igxHeaderFooter.Text = "Center Header"
```

See Also HeaderFooter object

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

FitToPagesAcross Property

Syntax *PageLayout*.**FitToPagesAcross**

Data Type Long (read/write)

Description The FitToPagesAcross property specifies the number of pages across (in the horizontal direction) to use when printing a diagram. A diagram has an arbitrary size, and could be larger (or smaller) than the size of a single page. This method makes a diagram print within the fixed number of pages across specified by this method. This is the same as changing the Fit To Pages Across option on the Page tab of the Page Setup dialog.

This property only works when the ScalingMode is set to ixScalingModeFitToPages.

Example The following example shows how to set the FitToPagesAcross and FitToPagesDown properties of the PageLayout object for the ActiveDiagram by using a With statement.

```
' Dimension the variables
Dim igxTGObj As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxGraphicBuilder As New GraphicBuilder
Dim igxDiagramObject As DiagramObject
Dim igxShadowFormat As ShadowFormat
' Create shape in the active diagram
For iCount = 1 To 5
    ' Make a horizontal row
    If iCount > 1 Then
        Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
            (1440 * (iCount * 1.5), 1440, _
            Application.ShapeLibraries.Item(1).Item(1))
    Else
        Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
            (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
    End If
    ' Make a vertical column
    If iCount > 1 Then
        Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
            (1440, 1440 * (iCount * 2), _
            Application.ShapeLibraries.Item(1).Item(1))
    End If
Next iCount
MsgBox "View the diagram"
With Application.ActiveDiagram.PageLayout
    .ScalingMode = ixScalingModeZoom
    ' Set the fit to pages across and down to 1
    .FitToPagesAcross = 1
    ' Set the fit to pages down and down to 1
    .FitToPagesDown = 1
End With
MsgBox "Fit to 1 page set. Now do a Print Preview on the diagram"
```

See Also FitToPagesDown property

ScalingMode property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

FitToPagesDown Property

Syntax *PageLayout*.**FitToPagesDown**

Data Type Long (read/write)

Description The FitToPagesDown property specifies the number of pages down (in the vertical direction) to use when printing a diagram. A diagram has an arbitrary size, and could be larger (or smaller) than the size of a single page. This method makes a diagram print within the fixed number of pages down specified by this method. This is the same as changing the fit to pages across on the Page tab of the Page Setup dialog.

This property only works when the ScalingMode is set to ixScalingModeFitToPages.

Example The following example shows how to set the FitToPagesAcross and FitToPagesDown properties of the PageLayout object for the ActiveDiagram by using a With statement.

```
' Dimension the variables
Dim igxTGObj As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxGraphicBuilder As New GraphicBuilder
Dim igxDiagramObject As DiagramObject
Dim igxShadowFormat As ShadowFormat
' Create shape in the active diagram
For iCount = 1 To 5
    ' Make a horizontal row
    If iCount > 1 Then
        Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
            (1440 * (iCount * 1.5), 1440, _
            Application.ShapeLibraries.Item(1).Item(1))
    Else
        Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
            (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
    End If
    ' Make a vertical column
    If iCount > 1 Then
        Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
            (1440, 1440 * (iCount * 2), _
            Application.ShapeLibraries.Item(1).Item(1))
    End If
Next iCount
MsgBox "View the diagram"
With Application.ActiveDiagram.PageLayout
    .ScalingMode = ixScalingModeZoom
    ' Set the fit to pages across and down to 1
    .FitToPagesAcross = 1
    ' Set the fit to pages down and down to 1
    .FitToPagesDown = 1
End With
MsgBox "Fit to 1 page set. Now do a Print Preview on the diagram"
```

See Also FitToPagesAcross property

ScalingMode property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

FooterHeight Property

Syntax *PageLayout*.**FooterHeight**

Data Type Long (read-only)

Description The FooterHeight property returns the height of the page footer section for all pages of a diagram. The value of this property is given in twips (1440 twips = 1.0 inch).

The page footer is where you can specify information that needs to be included on every page of a diagram. This information only shows up when the diagram is printed.

Example The following example retrieves the height of the page footer section in twips.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
' Set the PageLayout variable
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the text for the Footer
igxPageLayout.CenterFooter.Text = "Center Footer"
' Display the height of the Footer in twips
MsgBox "The Footer height is " & igxPageLayout.FooterHeight
```

See Also HeaderHeight property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

HeaderHeight Property

Syntax *PageLayout*.**HeaderHeight**

Data Type Long (read-only)

Description The HeaderHeight property returns the height of the page header section for all pages of a diagram. The value of this property is given in twips (1440 twips = 1.0 inch).

The page header is where you can specify information that needs to be included on every page of a diagram. This information only shows up when the diagram is printed.

Example The following example retrieves the height of the page header section in twips.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
' Set the PageLayout variable
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the text for the Header
igxPageLayout.CenterHeader.Text = "Center Header"
' Display the height of the Header in twips
MsgBox "The Header height is " & igxPageLayout.HeaderHeight
```

See Also FooterHeight property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

LeftFooter Property

Syntax *PageLayout*.**LeftFooter**

Data Type HeaderFooter object (read-only, See Object Properties)

Description The LeftFooter property returns a HeaderFooter object that is used to control the properties and format for the left region of the page footer. The specified properties and formatting are applied to every page of the diagram. The HeaderFooter object controls the same behavior that can be found in the Header/Footer tab of the Page Setup dialog.

Example The following example gets the HeaderFooter object for the active diagram and sets the left Footer text.

```
' Dimension the variables
Dim igxHeaderFooter As HeaderFooter
' Retrieve the HeaderFooter object using the LeftFooter property
Set igxHeaderFooter = ActiveDiagram.PageLayout.LeftFooter
' Set the text for the left Footer
igxHeaderFooter.Text = "Left Footer Text"
```

See Also HeaderFooter object

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

LeftHeader Property

Syntax *PageLayout*.**LeftHeader**

Data Type HeaderFooter object (read-only, See Object Properties)

Description The LeftHeader property returns a HeaderFooter object that is used to control the properties and format for the left region of the page header. The specified properties and formatting are applied to every page of the diagram. The HeaderFooter object controls the same behavior that can be found in the Header/Footer tab of the Page Setup dialog.

Example The following example gets the HeaderFooter object for the active diagram and sets the left Header text.

```
' Dimension the variables
Dim igxHeaderFooter As HeaderFooter
' Retrieve the HeaderFooter object using the LeftHeader property
Set igxHeaderFooter = ActiveDiagram.PageLayout.LeftHeader
' Set the text for the left header
igxHeaderFooter.Text = "Left Header Text"
```

See Also HeaderFooter object

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

LeftMargin Property

Syntax *PageLayout*.**LeftMargin**

Data Type Long (read/write)

Description The LeftMargin property specifies the size of the left margin for all pages that comprise a diagram. This property provides the same functionality as the Page Setup option—Margins tab—Left Margin control through the user interface. The value of this property is specified in twips (1440 twips = 1.0 inch).

Example The following example shows how to set the margin properties of the PageLayout object for the ActiveDiagram by using a With statement. It sets the bottom and top margins to ½ inch, and the right and left margins to ¼ inch.

```
With Application.ActiveDiagram.PageLayout
    ' Set the top and bottom margins to 1/2 inch
    .BottomMargin = 1440 / 2
    .TopMargin = 1440 / 2
    ' Set the left and right margins to 1/4 inch
    .RightMargin = 1440 / 4
    .LeftMargin = 1440 / 4
End With
```

See Also BottomMargin property

RightMargin property

TopMargin property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

Orientation Property

Syntax *PageLayout*.**Orientation**

Data Type IxPageOrientation enumerated constant (read/write)

Description The Orientation property specifies whether the pages for a diagram use portrait or landscape orientation. Portrait means that the longest dimension of the page is the vertical dimension; landscape means that the longest dimension of the page is the horizontal dimension. This property provides the same functionality as using the options on the Page Setup—Page tab through the interface.

The IxPageOrientation constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant
0	ixPrintPortrait
1	ixPrintLandscape

Example The following example shows how to set the Orientation property of the PageLayout object for the ActiveDiagram by using a With statement. The orientation is set to “Landscape”.

```
With Application.ActiveDiagram.PageLayout
    ' Set the page layout to landscape
    .Orientation = ixPageLandscape
End With
```

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

OverlapAmount Property

Syntax *PageLayout*.**OverlapAmount**

Data Type Long (read/write)

Description The OverlapAmount property specifies the overlap amount of a multi-page printed diagram. When the PageTitleMode is set to ixPerDiagram, the OverlapAmount is how much duplicate is printed on the right and bottom area of the page. For example, if the OverlapAmount is set to ½", then the last ½" of page one and the first ½" of page two are exactly the same. This allows the user to cut out the margins and paste the pages together into one large diagram. This is the same as setting the overlap amount in the Options tab under Page Setup.

Example The following example is sets the overlap amount in the ActiveDiagram using the PageLayout object.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the overlap amount to 1/2 inch.
igxPageLayout.OverlapAmount = 1440 / 2
```

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

PageCount Property

Syntax *PageLayout*.**PageCount**

Data Type Long (read-only)

Description The PageCount property returns the number of pages required to print the diagram.

Example The following example returns the page count, and displays it in a message box.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Display the current page count
MsgBox " The page count is " & igxPageLayout.PageCount
```

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

PageHeight Property

Syntax *PageLayout*.**PageHeight**

Data Type Long (read/write)

Description The PageHeight property specifies the page height (the vertical dimension of the page). This property always sets the vertical dimension, no matter which orientation (portrait or landscape) is used. The value of this property is specified in twips (1440 twips = 1.0 inch). This property provides the same functionality as the Page Height control on the Page tab, in the Page Setup dialog.

Example The following example gets the ActiveDiagram, and sets the page height to 14 inches.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the page height to 14 inches
igxPageLayout.PageHeight = 1440 * 14
```

See Also PageWidth property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

PageOrder Property

Syntax *PageLayout*.**PageOrder**

Data Type IxPageOrder enumerated constant (read/write)

Description The PageOrder property specifies how page ordering should be handled when a diagram fits onto multiple pages. This property controls the order in which pages are numbered. That is, if a diagram is larger than one page, is page number 2 below or to the right of page number 1? Page number 1 is always oriented at the top, left corner of the diagram. This is the same as setting the page order on the Options tab under Page Setup.

The IxPageOrder constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixDownThenAcross
1	ixAcrossThenDown

Example The following example gets the ActiveDiagram, and sets the page order to ixAcrossThenDown.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the page order to across then down
igxPageLayout.PageOrder = ixAcrossThenDown
```

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

PageTitleMode Property

Syntax *PageLayout*.**PageTitleMode**

Data Type IxPageTitleMode enumerated constant (read/write)

Description The PageTitleMode property specifies how the page headers are handled. When PageTitleMode is set to ixPerPage, then the headers and footers are printed on each page of output. When PageTitleMode is set to ixPerDiagram, then the headers and footers are printed once for the entire diagram.

Note that the OverlapAmount property is only used when this property is set to ixPerDiagram.

The IxPageOrder constant defines valid values for this property, which are listed in the following table.

Value	Name of Constant
0	IxPerPage
1	IxPerDiagram

Example The following example gets the ActiveDiagram, and sets the page title mode to ixPerDiagram.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the page title mode to "per diagram"
igxPageLayout.PageTitleMode = ixPerDiagram
```

See Also OverlapAmount property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

PageWidth Property

Syntax *PageLayout*.**PageWidth**

Data Type Long (read/write)

Description The PageWidth property specifies the page width (the horizontal dimension of the page). This property always sets the horizontal dimension, no matter which orientation (portrait or landscape) is used. The value of this property is specified in twips (1440 twips = 1.0 inch). This property provides the same functionality as the Page Width control on the Page tab, in the Page Setup dialog.

Example The following example gets the ActiveDiagram, and sets the page width to 11 inches.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the page width to 11 inches
igxPageLayout.PageWidth = 1440 * 11
```

See Also PageHeight property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

PaperSize Property

Syntax *PageLayout*.**PaperSize**

Data Type IxPaperSize enumerated constant (read/write)

Description The PaperSize property specifies the size of the paper to use for printing an iGrafx Professional diagram or component.

The IxPaperSize constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixPaperSizeLetter
1	ixPaperSizeLegal
2	ixPaperSizeTabloid
3	ixPaperSizeExecutive
4	ixPaperSizeCSheet
5	ixPaperSizeDSheet
6	ixPaperSizeESheet
7	ixPaperSizeLedger
8	ixPaperSizeStatement
9	ixPaperSizeFolio
10	ixPaperSize10x14
11	ixPaperSizeA5
12	ixPaperSizeA4
13	ixPaperSizeA3
14	ixPaperSizeA2
15	ixPaperSizeA1
16	ixPaperSizeA0
17	ixPaperSizeB5
18	ixPaperSizeB4
19	ixPaperSizeQuarto
20	ixPaperSizeEuroFanFold
21	ixPaperSizeCustom

Example The following example gets the ActiveDiagram, and sets the paper size to folio.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the paper size to folio
igxPageLayout.PaperSize = ixPaperSizeFolio
```

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

PrintFrames Property

Syntax *PageLayout*.**PrintFrames**[= {**True | False**}]

Data Type Boolean (read/write)

Description The PrintFrames property specifies whether to include a border that highlights the margins of each page when printing a diagram. This property provides the same functionality as selecting the print frames checkbox in the Options tab under Page Setup.

Example The following example gets the ActiveDiagram, and then sets the PrintFrames property to True.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Turn the print frames option on
igxPageLayout.PrintFrames = True
```

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

PrintNotes Property

Syntax *PageLayout*.**PrintNotes**[= {**True | False**}]

Data Type Boolean (read/write)

Description The PrintNotes property specifies whether to print any notes that are part of the diagram. This property provides the same functionality as the Print Notes checkbox in the Options tab in the Page Setup dialog.

Example The following example gets the ActiveDiagram, and turns on the Print Notes option for the diagram.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Turn the print notes option on
igxPageLayout.PrintNotes = True
```

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

RightFooter Property

Syntax *PageLayout*.**RightFooter**

Data Type HeaderFooter object (read-only, See Object Properties)

Description The RightFooter property returns a HeaderFooter object that is used to control the properties and format for the right region of the page footer. The specified properties and formatting are applied to every page of the diagram. The HeaderFooter object controls the same behavior that can be found in the Header/Footer tab of the Page Setup dialog.

Example The following example gets the HeaderFooter object for the active diagram and sets the right Footer text.

```
' Dimension the variables
Dim igxHeaderFooter As HeaderFooter
' Retrieve the HeaderFooter object using the RightFooter property
Set igxHeaderFooter = ActiveDiagram.PageLayout.RightFooter
' Set the text for the right Footer
igxHeaderFooter.Text = "Right Footer"
```

See Also HeaderFooter object

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

RightHeader Property

Syntax *PageLayout*.**RightHeader**

Data Type HeaderFooter object (read-only, See Object Properties)

Description The RightHeader property returns a HeaderFooter object that is used to control the properties and format for the right region of the page header. The specified properties and formatting are applied to every page of the diagram. The HeaderFooter object controls the same behavior that can be found in the Header/Footer tab of the Page Setup dialog.

Example The following example gets the HeaderFooter object for the active diagram and sets the right Header text.

```
' Dimension the variables
Dim igxHeaderFooter As HeaderFooter
' Retrieve the HeaderFooter object using the RightHeader property
Set igxHeaderFooter = ActiveDiagram.PageLayout.RightHeader
' Set the text for the right Header
igxHeaderFooter.Text = "Right Header"
```

See Also HeaderFooter object

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

RightMargin Property

Syntax *PageLayout*.**RightMargin**

Data Type Long (read/write)

Description The RightMargin property specifies the size of the right margin for all pages that comprise a diagram. This property provides the same functionality as the Page Setup option—Margins tab—Right Margin control through the user interface. The value of this property is specified in twips (1440 twips = 1.0 inch).

Example The following example shows how to set the margin properties of the PageLayout object for the ActiveDiagram by using a With statement. It sets the bottom and top margins to ½ inch, and the right and left margins to ¼ inch.

```
With Application.ActiveDiagram.PageLayout
    ' Set the top and bottom margins to 1/2 inch
    .BottomMargin = 1440 / 2
    .TopMargin = 1440 / 2
    ' Set the left and right margins to 1/4 inch
    .RightMargin = 1440 / 4
    .LeftMargin = 1440 / 4
End With
```

See Also BottomMargin property

LeftMargin property

TopMargin property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

ScalingMode Property

Syntax *PageLayout*.**ScalingMode**

Data Type IxScalingMode enumerated constant (read/write)

Description The ScalingMode property specifies how to scale a diagram or a component. Either object can be scaled by a percentage, or can be fit to a specified number of pages. This property provides the same functionality as the Scaling Options section of Page tab in the Page Setup dialog.

If this property is set to ixScalingModeFitToPages, then use the FitToPagesAcross and FitToPagesDown properties to set the number of pages wide and tall.

If this property is set to ixScalingModeZoom, use the Zoom property to set the zoom percentage at which you want to PRINT or DISPLAY the diagram or component.

The IxScalingMode constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixScalingModeFitToPages
1	ixScalingModeZoom

Example The following example gets the ActiveDiagram, and sets the scaling mode to fit to pages.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the scaling mode to fit to pages
igxPageLayout.ScalingMode = ixScalingModeFitToPages
```

See Also FitToPagesAcross property

FitToPagesDown property

Zoom property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

TopMargin Property

Syntax *PageLayout*.**TopMargin**

Data Type Long (read/write)

Description The TopMargin property specifies the size of the top margin for all pages that comprise a diagram. This property provides the same functionality as the Page Setup option—Margins tab—Top Margin control through the user interface. The value of this property is specified in twips (1440 twips = 1.0 inch).

Example The following example shows how to set the margin properties of the PageLayout object for the ActiveDiagram by using a With statement. It sets the bottom and top margins to ½ inch, and the right and left margins to ¼ inch.

```
With Application.ActiveDiagram.PageLayout
    ' Set the top and bottom margins to 1/2 inch
    .BottomMargin = 1440 / 2
    .TopMargin = 1440 / 2
    ' Set the left and right margins to 1/4 inch
    .RightMargin = 1440 / 4
    .LeftMargin = 1440 / 4
End With
```

See Also BottomMargin property

LeftMargin property

RightMargin property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

Zoom Property

Syntax *PageLayout*.**Zoom**

Data Type Double (read/write)

Description The Zoom property specifies the zoom scale to use for printing a diagram or component. The property affects the display of the diagram in the iGrafx Professional interface, and several other PageLayout properties as follows:

- The page separator lines in the diagram display adjust to show the page size for the current value of the Zoom property.
- The values of any page width or height properties are recalulated based on the zoom factor.

For instance, if you have an 8.5” x 11” page with 3/4” margins, your page size as shown in the interface is 7” x 9.5”. These values can be verified with the PageWidth and PageHeight properties. If you then set the Zoom property to 0.5 (50% zoom factor), your page size becomes 14” x 19”. However, in the diagram display, and when the diagram is printed, the 14” x 19” area is still one page, and will print correctly on an 8.5” x 11” piece of paper.

The Zoom property is used only if the ScalingMode property is set to ixScalingModeZoom. It provides the same functionality as setting the ‘Scaling, Adjust To’ value on the Page tab of the Page Setup dialog.

Example The following example has two parts. The first part, below, shows how the display and printing of the diagram are changed when the zoom scale is set at 50%. Compare this code to the code in Part 2, which leaves the zoom scale at 100%. The sample sets up page layout properties for the active diagram, and displays some information about those property settings. The zoom scale is changed from 100% to 50%, and then a row and column of shapes are added to the diagram. At a 50% zoom scale, all the shapes fit on one page, and this is verified in the Print Preview window.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPage As Page
Dim iCount As Integer
Dim igxPageLayout As PageLayout
' Get the PageLayout object
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the diagram's Page Layout and view properties
ActiveDiagram.Views.Item(1).DiagramView.Width = 1440 * 8
igxPageLayout.PageOrder = ixAcrossThenDown
igxPageLayout.OverlapAmount = 0
igxPageLayout.PaperSize = ixPaperSizeLetter
igxPageLayout.ScalingMode = ixScalingModeZoom
igxPageLayout.Zoom = 1
MsgBox "View the diagram. Page layout zoom factor is 100%"
' Display page size information
MsgBox "Page size is " _
    & CSng(ActiveDiagram.Pages.Item(1).Width / 1440) _
    & " x " & CSng(ActiveDiagram.Pages.Item(1).Height / 1440) _
    & Chr$(13) & "Page overlap is " _
    & CSng(ActiveDiagram.PageLayout.OverlapAmount / 1440)
' Adjust the width of the diagram view
ActiveDiagram.Views.Item(1).DiagramView.Width = 1440 * 17
MsgBox "Adjusted the view's width to 17 inches."
' Set the Page layout zoom factor to 50%
igxPageLayout.Zoom = 0.5
```

```
MsgBox "View the diagram. Page layout zoom factor set to 50%"
MsgBox "Page size is " _
    & CSng(ActiveDiagram.Pages.Item(1).Width / 1440) _
    & " x " & CSng(ActiveDiagram.Pages.Item(1).Height / 1440) _
    & Chr$(13) & "Page overlap is " _
    & CSng(ActiveDiagram.PageLayout.OverlapAmount / 1440)
' Create seven shapes in the active diagram in a row
For iCount = 1 To 7
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * iCount, 1440, _
        Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
        ActiveDiagram.DiagramObjects(iCount).Left = _
            ActiveDiagram.DiagramObjects(iCount - 1).Right + 980
    End If
Next iCount
MsgBox "Added a row of shapes."
' Add seven more shapes in a column
igxCurrentDOCount = ActiveDiagram.DiagramObjects.Count
For iCount = 1 To 7
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (ActiveDiagram.DiagramObjects(2).CenterX, _
        1440 * (iCount + 1), _
       Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
       ActiveDiagram.DiagramObjects(igxCurrentDOCount + iCount) _
            .Top = ActiveDiagram.DiagramObjects((igxCurrentDOCount _
            + iCount) - 1).Bottom + 980
    End If
Next iCount
MsgBox "Added a column of shapes."
' Print preview the diagram
ActiveDiagram.Views(1).DiagramView.PrintPreview
```

Part 2

Run the following code to see how the diagram is printed if the zoom scale is at 100%. Now when the shapes are added, they no longer fit onto one page; instead, the diagram now requires four pages to ft the shapes. This is verified in the Print Preview window.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPage As Page
Dim iCount As Integer
Dim igxPageLayout As PageLayout
' Get the PageLayout object
Set igxPageLayout = ActiveDiagram.PageLayout
' Set the diagram's Page Layout and view properties
ActiveDiagram.Views.Item(1).DiagramView.Width = 1440 * 8
igxPageLayout.PageOrder = ixAcrossThenDown
igxPageLayout.OverlapAmount = 0
igxPageLayout.PaperSize = ixPaperSizeLetter
igxPageLayout.ScalingMode = ixScalingModeZoom
igxPageLayout.Zoom = 1
MsgBox "View the diagram. Page layout zoom factor is 100%"
```

```
' Display page size information
MsgBox "Page size is " _
    & CSng(ActiveDiagram.Pages.Item(1).Width / 1440) _
    & " x " & CSng(ActiveDiagram.Pages.Item(1).Height / 1440) _
    & Chr$(13) & "Page overlap is " _
    & CSng(ActiveDiagram.PageLayout.OverlapAmount / 1440)
' Adjust the width of the diagram view
ActiveDiagram.Views.Item(1).DiagramView.Width = 1440 * 17
MsgBox "Adjusted the view's width to 17 inches."
' Create seven shapes in the active diagram in a row
For iCount = 1 To 7
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * iCount, 1440, _
        Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
        ActiveDiagram.DiagramObjects(iCount).Left = _
            ActiveDiagram.DiagramObjects(iCount - 1).Right + 980
    End If
Next iCount
MsgBox "Added a row of shapes."
' Add seven more shapes in a column
igxCurrentDOCount = ActiveDiagram.DiagramObjects.Count
For iCount = 1 To 7
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (ActiveDiagram.DiagramObjects(2).CenterX, _
        1440 * (iCount + 1), _
       Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
       ActiveDiagram.DiagramObjects(igxCurrentDOCount + iCount) _
            .Top = ActiveDiagram.DiagramObjects((igxCurrentDOCount _
            + iCount) - 1).Bottom + 980
    End If
Next iCount
MsgBox "Added a column of shapes."
' Print preview the diagram
ActiveDiagram.Views(1).DiagramView.PrintPreview
```

See Also ScalingMode property

{button PageLayout object,JI(`igrafxrf.HLP',`PageLayout_Object')}

Paragraph Object

The Paragraph object represents a paragraph of text within the Paragraphs collection. Individual Paragraph objects are accessed through the Paragraphs collection by using the Item method. A carriage return marks the end of one paragraph and the start of the next. For example, two bullet list items would be two separate Paragraph objects. In VBA, you can use the vbCr constant to specify a carriage return. For example, the following code creates four paragraphs in the shape "MyShape".

```
MyShape.Text = "This" + vbCr + "is" + vbCr + "a" + vbCr + "Test."
```

To access the actual text of a paragraph, you can use either the Text or the TextLF property. Both properties return the entire text string for the paragraph. The difference is that the TextLF property preserves any carriage returns in the string, while the Text property replaces carriage returns with spaces. The Text property is more useful for parsing the text for specific sequences, etc., while the TextLF property is more useful for operations such as adding or removing paragraphs. See the Text and TextLF properties for details on their use.

You can add a new paragraph by getting the TextLF property, inserting a carriage return at the end of the string, and then adding more text. For example:

```
MyShape.TextLF = MyShape.TextLF + vbCr + "New paragraph"
```

Each Paragraph object has its own ParagraphFormat object that controls how the text of the paragraph is formatted. Therefore, two paragraphs can have different formatting, even though they are in the same TextBlock.

The Paragraph object is used by the following objects:

- TextBlock
- ChildTextBlock
- TextGraphicObject object

Properties, Methods, and Events

All of the properties, methods, and events for the Paragraph object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Indent	
ParagraphFormat	Outdent	
Parent		
Text		
TextLF		
TextRange		

Related Topics

Paragraphs object

iGrafx API Object Hierarchy

Indent Method

Syntax *Paragraph*.**Indent**

Description The Indent method indents the text of the specified paragraph from the left edge of the text block. The BlockFormat.TabWidth property controls the amount of indentation. For example, if BlockFormat.TabWidth is set to ¼ inch (360 twips), then invoking the Indent method indents the paragraph ¼ inch from the left edge of the text block (or its previous position). The Indent method can be applied more than once—each time it indents the paragraph by another TabWidth amount.

Example The following example creates a shape and adds some text within it. The text is then indented.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxParagraph As Paragraph
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the text of the shape
igxShape.Text = "This text will be indented"
MsgBox "View the diagram"
' Get the Paragraph object from the shape's TextBlock
Set igxParagraph = igxShape.TextBlock.Paragraphs.Item(1)
' Set the text to be indented right by the space of one Tab
igxParagraph.Indent
MsgBox "View the diagram"
```

See Also Outdent method

BlockFormat.TabWidth property

{button Paragraph object,JI(`igrafxrf.HLP',`Paragraph_Object')}

Outdent Method

Syntax *Paragraph*.**Outdent**

Description The Outdent method removes an indent from the specified paragraph. The BlockFormat.TabWidth property controls the amount of space the text is moved towards the left edge of the TextBlock. For example, if BlockFormat.TabWidth is set to ¼ inch (360 twips), then invoking the Outdent method moves the paragraph ¼ inch towards the left edge of the TextBlock.

Note that the purpose of the Outdent method is to undo the effect of the Indent method. Applying the Outdent method when the text is at the left edge of the text block (that is, it has not been indented) has no effect.

Example The following example creates a shape and adds some text within it. The text is then indented.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxParagraph As Paragraph
' Set igxShape variable to the Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the text of the shape
igxShape.Text = "This text will be indented"
MsgBox "View the diagram"
' Get the Paragraph object from the shape's TextBlock
Set igxParagraph = igxShape.TextBlock.Paragraphs.Item(1)
' Set the text to be indented right by the space of one Tab
igxParagraph.Indent
MsgBox "View the diagram"
' Reset the text to be aligned left
igxParagraph.Outdent
MsgBox "View the diagram"
```

See Also Indent method

BlockFormat.TabWidth property

{button Paragraph object,JI(`igrafxrf.HLP',`Paragraph_Object')}

ParagraphFormat Property

Syntax *Paragraph*.**ParagraphFormat**

Data Type ParagraphFormat object (read-only, See Object Properties)

Description The ParagraphFormat property returns the ParagraphFormat object for the specified Paragraph object. Each Paragraph object has its own ParagraphFormat object that controls how the text of the paragraph is formatted.

Example The following example creates a text block, returns the paragraph format, and sets a circular bullet type.

```
' Dimension the variables
Dim igxTextGraphicObject As TextGraphicObject
Dim igxParagraph As Paragraph
Dim igxParagraphFormat As ParagraphFormat
' Create a TextGraphic object in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440, , , "A string of text")
MsgBox "View the diagram"
' Set igxParagraph variable to the first Paragraph object
Set igxParagraph = igxTextGraphicObject.Paragraphs.Item(1)
' Set the bullet type for the paragraph
igxParagraph.ParagraphFormat.BulletType = ixBulletCircle
MsgBox "View the diagram"
```

See Also ParagraphFormat object

iGrafx API Object Hierarchy

{button Paragraph object,JI(`igrafxrf.HLP',`Paragraph_Object')}

TextRange Property

Syntax *Paragraph*.**TextRange**([*First* As Long = 1], [*Last* As Long])

Data Type TextRange object (read-only, See Object Properties)

Description The TextRange property returns a TextRange object from the specified Paragraph object. The purpose of this property is to provide control over a range of text within a paragraph.

The TextRange object lets you work with a range of text. The *First* and *Last* arguments specify the start and end positions of the text range. For example, specifying Paragraph1.TextRange(1,5) returns a TextRange that contains the first five characters of the paragraph. Specifying the property without providing the *First* and *Last* arguments returns a TextRange with all the characters in the paragraph. The *First* argument defaults to a value of 1, so to select from the first character of the paragraph only requires specifying the last character.

Example The following example creates a TextGraphicObject in the diagram with a text string. Then part of the first paragraph is extracted into a text range. The text range is then used in the specification of a second paragraph.

```
' Dimension the variables
Dim igxTextGraphicObj As TextGraphicObject
Dim igxParagraph As Paragraph
Dim igxTextRange As TextRange
' Create a TextGraphic in the active diagram
Set igxTextGraphicObj = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440, , , "A string of text")
MsgBox "Created a TextGraphic object"
' Set igxParagraph variable to the first Paragraph object
Set igxParagraph = igxTextGraphicObj.Paragraphs.Item(1)
' Get the TextRange object
Set igxTextRange = igxParagraph.TextRange(3, 8)
' Display the text within the TextRange object
MsgBox "The text in the TextRange object is '" _
    & igxTextRange.Text & "'"
' Create a second paragraph, inserting the TextRange text
igxParagraph.TextLF = igxParagraph.Text & Chr$(13) _
    & "New text with a " & igxTextRange.Text
MsgBox "There are now " & igxTextGraphicObj.Paragraphs.Count _
    & " paragraphs."
```

See Also TextRange object

iGrafx API Object Hierarchy

{button Paragraph object,JI(`igrafxrf.HLP',`Paragraph_Object')}

Paragraphs Object

The Paragraphs collection is a collection of Paragraph objects. This object provides access to Paragraph objects that have been created for the following objects:

- Note
- HeaderFooter
- TextBlock (which is associated with Shape and ShapeClass)
- ChildTextBlock (which is associated with a TextBlock)
- Department
- TextGraphicObject

The Paragraphs collection provides the following functionality for working with Paragraph objects.

- The ability to access individual Paragraph objects.
- The ability to determine how many Paragraph objects are currently in the collection.

You can add and delete paragraphs from the collection by changing either the Text or TextLF property of the object associated with the Paragraphs collection.

Properties, Methods, and Events

All of the properties, methods, and events for the Paragraphs object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
Parent		

Related Topics

Paragraph object

iGrafx API Object Hierarchy

Item Method

Syntax *Paragraphs*.**Item**(*Index* As Integer) As Paragraph

Description The Item method returns the Paragraph object at the specified *Index* from the Paragraphs collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Paragraph. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example sets the alignment for odd numbered paragraphs to left alignment, and center-aligns the even numbered paragraphs.

```
' Dimension the variables
Dim igxParas As Paragraphs
Dim iCount As Integer
Set igxParas = ActiveDiagram.DiagramObjects(1) _
    .Shape.TextBlock.Paragraphs
For iCount = 1 To igxParas.Count
   If (iCount Mod 2 = 0) Then
       igxParas.Item(iCount).ParagraphFormat.Alignment = _
           ixHorizontalAlignCenter
    Else
       igxParas.Item(iCount).ParagraphFormat.Alignment = _
           ixHorizontalAlignLeft
    End If
Next iCount
```

{button Paragraphs object,JI(`igrafxrf.HLP',`Paragraphs_Object')}

TextBlock Object

The TextBlock object is a container for text. In addition, the TextBlock object provides for additional text areas, called ChildTextBlock objects. A TextBlock can have multiple ChildTextBlock objects associated with it. A ChildTextBlock takes space away from the main text block; that is, it is created inside the area allocated for the main text block.

The TextBlock object is associated with the Shape and ShapeClass objects. A shape has one text block; however, that text block can have many child text blocks. To format the text contained in the TextBlock and ChildTextBlock objects, you use the BlockFormat object.

To access the actual text in a TextBlock, you can use either the Text or TextLF property. Both properties return the entire text string for the TextBlock. The difference is that the TextLF property preserves any carriage returns in the string, while the Text property replaces carriage returns with spaces. However, when writing to either property, carriage returns are recognized. Refer to the documentation of these properties for complete information about their use.

The RelativePositionType, TopMargin, BottomMargin, LeftMargin, and RightMargin properties control the position and size of the text block within a shape. An important detail about these properties is that the TopMargin, BottomMargin, LeftMargin, and RightMargin properties work differently depending on the value of the RelativePositionType property. For more information, refer to these topics.

Note that a TextGraphicObject object is also a container for text. However, it does not have a TextBlock object like a shape.

Properties, Methods, and Events

All of the properties, methods, and events for the TextBlock object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
BlockFormat		
BottomMargin		
ChildTextBlocks		
LeftMargin		
Paragraphs		
Parent		
RelativePositionType		
RightMargin		
Text		
TextLF		
TextMargin		
TextRange		
TopMargin		

Related Topics

ChildTextBlock object

BlockFormat object

TextGraphicObject object

iGrafx API Object Hierarchy

BlockFormat Property

Syntax *TextBlock*.**BlockFormat**

Data Type BlockFormat object (read-only, See Object Properties)

Description The BlockFormat property returns the BlockFormat object associated with the specified TextBlock object.

The BlockFormat object controls the formatting of the text associated with a shape (the TextBlock or ChildTextBlock object). The TextBlock object (there is only one per shape) and all ChildTextBlock objects (there can be zero or more per TextBlock) have there own distinct BlockFormat objects for controlling text formatting.

Example The following example creates a shape with text in the active diagram, then utilizes the BlockFormat object's FillColor to change the fill color of the TextBlock object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlock As TextBlock
Dim igxBlockFormat As BlockFormat
' Create a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the text and color of the shape
igxShape.Text = "This text will be in the Text Block"
igxShape.FillColor = vbRed
' Get the TextBlock from the Shape object, set its margins
Set igxTextBlock = igxShape.TextBlock
Set igxTextBlock.LeftMargin = .1
Set igxTextBlock.RightMargin = .1
Set igxTextBlock.TopMargin = .1
Set igxTextBlock.BottomMargin = .1
' Set igxBlockFormat variable to the BlockFormat object
Set igxBlockFormat = igxTextBlock.BlockFormat
' Set the FillColor property to blue
igxBlockFormat.FillColor = vbBlue
MsgBox "View the diagram"
```

See Also BlockFormat object

iGrafx API Object Hierarchy

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

BottomMargin Property

Syntax *TextBlock*.**BottomMargin**

Data Type Double (read/write)

Description The BottomMargin property specifies how much of a margin to provide for the text block at the bottom edge of a shape. The units are either twips (1440 twips = 1 inch), or a percentage. Values can be positive or negative.

Setting this property value (that is, the units of measure) is dependent upon the value of the RelativePositionType property:

- If RelativePositionType is ixTextPositionFixed, then the value of this property is given as an absolute distance specified in twips (a BottomMargin of 144 is 1/10 inch). If the shape is resized, the margin stays at 1/10 inch.
- If RelativePositionType is ixTextPositionPercentage, then the value of this property is given as a percentage of the shape coordinate space (a BottomMargin of .1 would be 10% of the shape's height). If the shape is resized, the margin size changes to maintain the same percentage of the shape's height.

A negative property value positions the bottom edge of the text block outside of the shape's border. For example, if the RelativePositionType is ixTextPositionPercentage, the BottomMargin value is –.1 and the height of the shape is one inch, then the bottom of the text block would be 1/10 of an inch below the bottom border of the shape.

See the BlockFormat object for information of formatting the text in a text block.

Example The following example gives the TextBlock a bottom margin of 0.5 inch.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlock As TextBlock
' Create a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the text of the shape
igxShape.Text = "Bottom margin"
MsgBox "View the diagram"
' Get the TextBlock from the Shape
Set igxTextBlock = igxShape.TextBlock
' Set the relative position of the text block
igxTextBlock.RelativePositionType = ixTextPositionFixed
' Set the bottom margin to 0.5 inch
igxTextBlock.BottomMargin = 1440 / 2
MsgBox "View the diagram"
' Resize the shape
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
MsgBox "View the diagram"
```

See Also LeftMargin property

RightMargin property

TopMargin property

RelativePositionType property

BlockFormat object

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

ChildTextBlocks Property

Syntax *TextBlock*.**ChildTextBlocks**

Data Type ChildTextBlocks collection object (read-only, See Object Properties)

Description The ChildTextBlocks property returns the ChildTextBlocks collection for the specified TextBlock object. You use this object to add or remove a ChildTextBlock, or to access an existing ChildTextBlock.

Example The following example creates a ChildTextBlock object, then places text in the ChildTextBlock object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlock As TextBlock
Dim igxChildTextBlocks As ChildTextBlocks
Dim igxChildTextBlock As ChildTextBlock
' Create a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "View the diagram"
' Get the ChildTextBlocks object through the Shape's TextBlock object
Set igxChildTextBlocks = igxShape.TextBlock.ChildTextBlocks
' Create a ChildTextBlock object
Set igxChildTextBlock = igxChildTextBlocks.AddFixed(ixTextRight, 720)
' Place text in the ChildTextBlock object
igxChildTextBlock.Text = "Child Text Block"
' Give the child text block a border
igxChildTextBlock.BlockFormat.LineFormat.Style = ixLineNormal
igxChildTextBlock.BlockFormat.LineFormat.Color = vbRed
MsgBox "View the diagram"
```

See Also ChildTextBlock object

ChildTextBlocks object

iGrafx API Object Hierarchy

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

LeftMargin Property

Syntax *TextBlock*.**LeftMargin**

Data Type Double (read/write)

Description The LeftMargin property specifies how much of a margin to provide for the text block at the left edge of a shape. The units are either twips (1440 twips = 1 inch), or a percentage. Values can be positive or negative.

Setting this property value (that is, the units of measure) is dependent upon the value of the RelativePositionType property:

- If RelativePositionType is ixTextPositionFixed, then the value of this property is given as an absolute distance specified in twips (a LeftMargin of 144 is 1/10 inch). If the shape is resized, the margin stays at 1/10 inch.
- If RelativePositionType is ixTextPositionPercentage, then the value of this property is given as a percentage of the shape coordinate space (a LeftMargin of .1 would be 10% of the shape's width). If the shape is resized, the margin size changes to maintain the same percentage of the shape's width.

A negative property value positions the left edge of the text block outside of the shape's border. For example, if the RelativePositionType is ixTextPositionPercentage, the LeftMargin value is –.1 and the width of the shape is one inch, then the left side of the text block would be 1/10 of an inch to the left of the shape's left border.

See the BlockFormat object for information of formatting the text in a text block.

Example The following example positions the left margin of the TextBlock for the shape at 0.5 inch.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlock As TextBlock
' Create a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the text of the shape
igxShape.Text = "Left margin"
MsgBox "View the diagram"
' Get the TextBlock from the Shape
Set igxTextBlock = igxShape.TextBlock
' Set the relative position of the text block
igxTextBlock.RelativePositionType = ixTextPositionFixed
' Set the left margin to 0.5 inch
igxTextBlock.LeftMargin = 1440 / 2
MsgBox "View the diagram"
' Resize the shape
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
MsgBox "View the diagram"
```

See Also BottomMargin property

RightMargin property

TopMargin property

RelativePositionType property

BlockFormat object

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

Paragraphs Property

Syntax *TextBlock*.**Paragraphs**

Data Type Paragraphs collection object (read-only, See Object Properties)

Description The Paragraphs property returns the Paragraphs collection associated with the specified TextBlock object. The Paragraphs object, through the Item method, provides access to the individual Paragragh objects.

Example The following example displays the number of Paragraph objects for the shape by accessing the Count property of the Paragraphs collection object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxParagraphs As Paragraphs
' Create a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the text of the shape
igxShape.Text = "Shape Text" & Chr(13) & "Another paragraph"
' Get the Paragraphs collection of the shape's TextBlock
Set igxParagraphs = igxShape.TextBlock.Paragraphs
' Display the number of Paragraph objects for the shape
MsgBox "The number of Paragraphs for the shape is " _
    & igxParagraphs.Count
```

See Also Paragraph object

Paragraphs object

iGrafx API Object Hierarchy

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

RelativePositionType Property

Syntax *TextBlock*.**RelativePositionType**

Data Type IxTextPositionType enumerated constant (read/write)

Description The RelativePositionType property specifies the method for sizing a shape's text block margins. The property works in conjunction with the TopMargin, BottomMargin, LeftMargin, and RightMargin properties to control how margin values are specified.

- If RelativePositionType is ixTextPositionFixed, then the value of this property is given as an absolute distance specified in twips (a LeftMargin of 144 is 1/10 inch). If the shape is resized, the margin stays at 1/10 inch.
- If RelativePositionType is ixTextPositionPercentage, then the value of this property is given as a percentage of the shape coordinate space (a LeftMargin of .1 would be 10% of the shape's width). If the shape is resized, the margin size changes to maintain the same percentage of the shape's width.

The IxTextPositionType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixTextPositionFixed
1	ixTextPositionPercentage

For information about setting the values of the TopMargin, BottomMargin, LeftMargin, and RightMargin properties and their effect, refer to those topics.

Example The following example set the relative position type to percentage for the associated TextBlock object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlock As TextBlock
' Create a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the text of the shape
igxShape.Text = "All margins, relative"
MsgBox "View the diagram"
' Get the TextBlock from the Shape
Set igxTextBlock = igxShape.TextBlock
' Set the relative position of the text block
igxTextBlock.RelativePositionType = ixTextPositionPercentage
' Set the margins to 15% of the shape's height and width
igxTextBlock.BottomMargin = .15
igxTextBlock.LeftMargin = .15
igxTextBlock.RightMargin = .15
igxTextBlock.TopMargin = .15
MsgBox "View the diagram"
' Resize the shape
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
MsgBox "Shape resized"
```

See Also

BottomMargin property
LeftMargin property
RightMargin property
TopMargin property
BlockFormat object

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

RightMargin Property

Syntax *TextBlock*.**RightMargin**

Data Type Double (read/write)

Description The RightMargin property specifies how much of a margin to provide for the text block at the right edge of a shape. The units are either twips (1440 twips = 1 inch), or a percentage. Values can be positive or negative.

Setting this property value (that is, the units of measure) is dependent upon the value of the RelativePositionType property:

- If RelativePositionType is ixTextPositionFixed, then the value of this property is given as an absolute distance specified in twips (a RightMargin of 144 is 1/10 inch). If the shape is resized, the margin stays at 1/10 inch.
- If RelativePositionType is ixTextPositionPercentage, then the value of this property is given as a percentage of the shape coordinate space (a RightMargin of .1 would be 10% of the shape's width). If the shape is resized, the margin size changes to maintain the same percentage of the shape's width.

A negative property value positions the right edge of the text block outside of the shape's border. For example, if the RelativePositionType is ixTextPositionPercentage, the RightMargin value is –.1 and the width of the shape is one inch, then the right side of the text block would be 1/10 of an inch to the right of the shape's right border.

See the BlockFormat object for information of formatting the text in a text block.

Example The following example positions the right margin of the TextBlock for the shape at 0.5 inch.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlock As TextBlock
' Create a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the text of the shape
igxShape.Text = "Right margin"
MsgBox "View the diagram"
' Get the TextBlock from the Shape
Set igxTextBlock = igxShape.TextBlock
' Set the relative position of the text block
igxTextBlock.RelativePositionType = ixTextPositionFixed
' Set the right margin to 0.5 inch
igxTextBlock.RightMargin = 1440 / 2
MsgBox "View the diagram"
' Resize the shape
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
MsgBox "Shape resized"
```

See Also BottomMargin property

LeftMargin property

TopMargin property

RelativePositionType property

BlockFormat object

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

TextMargin Property

Syntax *TextBlock*.**TextMargin**

Data Type Integer (read/write)

Description The TextMargin property specifies the width of a border around the text area of a shape or TextGraphicObject. The TextMargin property is in addition to the left, right, top,and bottom margin properties, providing additional margin control for the text block. The units are in twips (1440 twips = 1 inch), and the default is 50. Values can be positive or negative.

A negative value positions the edges of the text block outside of the shape's border.

See the BlockFormat object for information of formatting the text in a text block.

Example The following example specifies a text margin of 1/5 an inch (288 twips) for the TextBlock of the shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Set all the shape's text block margins to 1/5 inch
igxShape.TextBlock.TextMargin = 288
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineDashed
    .Width = 1
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "This is the Main Text Block Area"
MsgBox "View the diagram"
```

See Also

BottomMargin property

LeftMargin property

RightMargin property

TopMargin property

RelativePositionType property

BlockFormat object

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

TextRange Property

Syntax *TextBlock*.**TextRange**([*First* As Long = 1], [*Last* As Long])

Data Type TextRange object (read-only, See Object Properties)

Description The TextRange property returns a TextRange object for the specified TextBlock object. The purpose of this property is to provide control over a range of text within a TextBlock.

The TextRange object lets you work with a range of text. The *First* and *Last* arguments specify the start and end positions of the text range. For example, specifying Paragraph1.TextRange(1,5) returns a TextRange that contains the first five characters of the paragraph. Specifying the property without providing the *First* and *Last* arguments returns a TextRange with all the characters in the paragraph. The *First* argument defaults to a value of 1, so to select from the first character of the paragraph only requires specifying the last character.

In addition, each Paragraph object contained within a TextBlock has its own TextRange object that can be used to select either all or part of the paragraph.

Example The following example gets the TextRange object for the TextBlock object and displays the text contents of the TextRange object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextRange As TextRange
' Create a shape in the active diagram.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxParagraph variable to the first Paragraph object.
igxShape.Text = "This is the text."
' Get the TextRange object of the shape's Text Block
Set igxTextRange = igxShape.TextBlock.TextRange(1, 3)
' Display the text within the TextRange object.
MsgBox "The text in the TextRange object is " & igxTextRange.Text
```

See Also TextRange object

Paragraph object

iGrafx API Object Hierarchy

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

TopMargin Property

Syntax *TextBlock*.**TopMargin**

Data Type Double (read/write)

Description The TopMargin property specifies how much of a margin to provide for the text block at the top edge of a shape. The units are either twips (1440 twips = 1 inch), or a percentage. Values can be positive or negative.

Setting this property value (that is, the units of measure) is dependent upon the value of the RelativePositionType property:

- If RelativePositionType is ixTextPositionFixed, then the value of this property is given as an absolute distance specified in twips (a TopMargin of 144 is 1/10 inch). If the shape is resized, the margin stays at 1/10 inch.
- If RelativePositionType is ixTextPositionPercentage, then the value of this property is given as a percentage of the shape coordinate space (a TopMargin of .1 would be 10% of the shape's height). If the shape is resized, the margin size changes to maintain the same percentage of the shape's height.

A negative property value positions the top edge of the text block outside of the shape's border. For example, if the RelativePositionType is ixTextPositionPercentage, the TopMargin value is –.1 and the height of the shape is one inch, then the top of the text block would be 1/10 of an inch above the top border of the shape.

See the BlockFormat object for information of formatting the text in a text block.

Example The following example positions the top margin of the TextBlock for the shape at 0.5 inch.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlock As TextBlock
' Create a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the text of the shape
igxShape.Text = "Top margin"
MsgBox "View the diagram"
' Get the TextBlock from the Shape
Set igxTextBlock = igxShape.TextBlock
' Set the relative position of the text block
igxTextBlock.RelativePositionType = ixTextPositionFixed
' Set the top margin to 0.5 inch
igxTextBlock.TopMargin = 1440 / 2
MsgBox "View the diagram"
' Resize the shape
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
MsgBox "Shape resized"
```

See Also

BottomMargin property

LeftMargin property

RightMargin property

RelativePositionType property

BlockFormat object

{button TextBlock object,JI(`igrafxrf.HLP',`TextBlock_Object')}

TextGraphicObject Object

The TextGraphicObject represents objects that are created with the text tool or one of the graphic tools. These objects are not shapes (but can be converted to shapes using the ConvertToShape method).

A TextGraphicObject differs from a Shape object in that it is not connectable by connector lines, and lacks many of the Shape object's features that apply to modeling. However, a TextGraphicObject can be attached to shapes and connector lines as a callout. A TextGraphicObject should be used to annotate (with text) or embellish (with graphics) diagrams that consist of Shapes and ConnectorLines.

The name of the TextGraphicObject object comes from the fact that it has a graphic component and a text component, much like a shape. When you use the Text tool to create a TextGraphicObject, it's graphic is initially blank. Likewise, when you use the Graphic tools to create a TextGraphicObject, it's text is initially blank, but can be filled in by typing while the TextGraphicObject is selected. Through the API's, you can create a TextGraphicObject, and add to, remove, or modify it's graphic or text elements in a variety of ways.

To access the graphical part of a TextGraphicObject, use its Graphic property. To access the actual text of a TextGraphicObject, you can use either the Text or the TextLF property. Both properties return the entire text string for the TextGraphicObject. The difference is that the TextLF property preserves any carriage returns in the string, while the Text property replaces carriage returns with spaces.

Properties, Methods, and Events

All of the properties, methods, and events for the TextGraphicObject object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	AttachTo	
BlockFormat	ConvertToShape	
CalloutLineDestinationArrowFormat	Detach	
CalloutLineFormat		
CalloutLineSourceArrowFormat		
DiagramObject		
FillFormat		
Graphic		
IsAttached		
LineFormat		
Paragraphs		
Parent		
ShadowFormat		
Text		
TextLF		
TextRange		
ThreeDFormat		

Related Topics

Graphic object

TextBlock object

Shape object

iGrafx API Object Hierarchy

AttachTo Method

Syntax *TextGraphicObject*.**AttachTo**(*ParentObject* As DiagramObject)

Description The AttachTo method attaches a TextGraphicObject to another object in the same diagram. The other object can be a connector line, a shape, or another TextGraphicObject. Once attached, you can cause a line to be drawn between the objects by using the CalloutLineFormat property. If you drag the object that is attached to the TextGraphicObject, the TextGraphicObject keeps its relative position to the attached object.

The *ParentObject* argument specifies the object to which to attach the TextGraphicObject.

Example The following example attaches a TextGraphicObject to a shape with a dashed blue callout line. The TextGraphicObject is then moved to two other locations.

```
' Dimension the variables
Dim igxTGObj As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a text graphic object in the active diagram
Set igxTGObj = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440 * 2, 1440 * 4, , , "A string of text")
' Set the properties of the callout line
Set igxLineFormat = igxTGObj.CalloutLineFormat
igxLineFormat.Color = vbBlue
igxLineFormat.Style = ixLineDashed
igxLineFormat.Width = 3
' Select the shape
Set igxDiagramObject = ActiveDiagram.DiagramObjects.Item(1)
' Attach the callout line from the text graphic object to the shape.
igxTGObj.AttachTo igxDiagramObject
' Move the text graphic to several locations
igxTGObj.DiagramObject.CenterX = 1440 * 4
MsgBox "View the diagram"
igxTGObj.DiagramObject.CenterY = 1440 * 2
MsgBox "View the diagram"
```

See Also Detach method

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

BlockFormat Property

Syntax *TextGraphicObject*.**BlockFormat**

Data Type BlockFormat object (read-only, See Object Properties)

Description The BlockFormat property returns the BlockFormat object associated with the specified TextGraphicObject object.

The BlockFormat object controls the formatting of the text associated with a TextGraphicObject object. Each TextGraphicObject has its own distinct BlockFormat object for controlling text formatting.

Example The following example retrieves the BlockFormat object of a TextGraphicObject and reorients it.

```
' Dimension the variables
Dim igxTextGraphicObject As TextGraphicObject
Dim igxBlockFormat As BlockFormat
' Create a TextGraphic in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440, , , "A string of text")
MsgBox "View the diagram"
Set igxBlockFormat = igxTextGraphicObject.BlockFormat
igxBlockFormat.Orientation = ixOrientation90
MsgBox "View the diagram"
```

See Also BlockFormat object

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

CalloutLineDestinationArrowFormat Property

Syntax *TextGraphicObject*.**CalloutLineDestinationArrowFormat**

Data Type ArrowFormat object (read-only, See Object Properties)

Description The CalloutLineDestinationArrowFormat property returns the ArrowFormat object that controls the callout line destination arrow for the specified TextGraphicObject.

Example The following example attaches a TextGraphicObject to a Shape and formats the callout line with source and destination arrowheads.

```
' Dimension the variables
Dim igxTextGraphicObject As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
Dim igxArrowFormat As ArrowFormat
Dim igxArrowFormat2 As ArrowFormat
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a text graphic object on the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440 * 2, 1440 * 4, , , "A string of text")
MsgBox "View the diagram"
' Set the properties of the callout line
Set igxLineFormat = igxTextGraphicObject.CalloutLineFormat
igxLineFormat.Width = 3
' Select the shape
Set igxDiagramObject = ActiveDiagram.DiagramObjects.Item(1)
' Attach the callout line from the text graphic object to the shape
igxTextGraphicObject.AttachTo igxDiagramObject
' Set the properties of the callout line destination arrow format
Set igxArrowFormat = _
   igxTextGraphicObject.CalloutLineDestinationArrowFormat
igxArrowFormat.Style = ixArrow12
igxArrowFormat.Color = vbGreen
MsgBox "View the diagram"
' Set the properties of the callout line source arrow format
Set igxArrowFormat2 = _
   igxTextGraphicObject.CalloutLineSourceArrowFormat
igxArrowFormat2.Style = ixArrow6
igxArrowFormat2.Color = vbRed
MsgBox "View the diagram"
```

See Also

ArrowFormat object

CalloutLineSourceArrowFormat property

CalloutLineFormat property

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

CalloutLineFormat Property

Syntax *TextGraphicObject*.**CalloutLineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The CalloutLineFormat property returns the LineFormat object that controls the callout line of the specified TextGraphicObject object.

Example The following example attaches a TextGraphicObject to a shape and formats the callout line.

```
' Dimension the variables
Dim igxTextGraphicObject As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
Dim igxArrowFormat As ArrowFormat
Dim igxArrowFormat2 As ArrowFormat
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a text graphic object in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440 * 2, 1440 * 4, , , "A string of text")
MsgBox "View the diagram"
' Set the properties of the callout line
Set igxLineFormat = igxTextGraphicObject.CalloutLineFormat
igxLineFormat.Color = vbBlue
igxLineFormat.Style = ixLineDashed
igxLineFormat.Width = 3
' Select the shape
Set igxDiagramObject = ActiveDiagram.DiagramObjects.Item(1)
' Attach the callout line from the text graphic object to the shape
igxTextGraphicObject.AttachTo igxDiagramObject
MsgBox "View the diagram"
```

See Also

LineFormat object

CalloutLineSourceArrowFormat property

CalloutLineDestinationArrowFormat property

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

CalloutLineSourceArrowFormat Property

Syntax *TextGraphicObject*.**CalloutLineSourceArrowFormat**

Data Type ArrowFormat object (read-only, See Object Properties)

Description The CalloutLineSourceArrowFormat property returns the ArrowFormat object that controls the callout line source arrow for the specified TextGraphicObject.

Example The following example attaches a TextGraphicObject to a Shape and formats the callout line with source and destination arrowheads.

```
' Dimension the variables
Dim igxTextGraphicObject As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
Dim igxArrowFormat As ArrowFormat
Dim igxArrowFormat2 As ArrowFormat
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a text graphic object on the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440 * 2, 1440 * 4, , , "A string of text")
MsgBox "View the diagram"
' Set the properties of the callout line
Set igxLineFormat = igxTextGraphicObject.CalloutLineFormat
igxLineFormat.Width = 3
' Select the shape
Set igxDiagramObject = ActiveDiagram.DiagramObjects.Item(1)
' Attach the callout line from the text graphic object to the shape
igxTextGraphicObject.AttachTo igxDiagramObject
' Set the properties of the callout line destination arrow format
Set igxArrowFormat = _
   igxTextGraphicObject.CalloutLineDestinationArrowFormat
igxArrowFormat.Style = ixArrow12
igxArrowFormat.Color = vbGreen
MsgBox "View the diagram"
' Set the properties of the callout line source arrow format
Set igxArrowFormat2 = _
   igxTextGraphicObject.CalloutLineSourceArrowFormat
igxArrowFormat2.Style = ixArrow6
igxArrowFormat2.Color = vbRed
MsgBox "View the diagram"
```

See Also ArrowFormat object

CalloutLineDestinationArrowFormat property

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

ConvertToShape Method

Syntax *TextGraphicObject*.**ConvertToShape**() As Shape

Description The ConvertToShape method is used to convert a TextGraphicObject into a Shape object. A Shape object has connectivity and can be used in modeling, whereas a TextGraphicObject cannot. When you convert a TextGraphicObject to a shape, you lose any attachments the TextGraphicObject previously had to shapes, connector lines, or other TextGraphicObject objects.

The result of the ConvertToShape method must be assigned to a variable of type “Shape”.

Example The following example turns a TextGraphicObject object into a Shape object.

```
' Dimension the variables
Dim igxTextGraphicObject As TextGraphicObject
Dim igxDiagramObj As DiagramObject
Dim igxShape As Shape
Dim igxLineFormat As LineFormat
' Create a TextGraphic in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440, , , "A string of text")
Set igxLineFormat = igxTextGraphicObject.LineFormat
igxLineFormat.Style = ixLineNormal
igxLineFormat.Color = vbBlue
MsgBox "View the diagram"
' Convert the TextGraphic to a Shape
Set igxShape = igxTextGraphicObject.ConvertToShape
' Test that the TextGraphic was actually converted to a shape
For Each igxDiagramObj In ActiveDiagram.DiagramObjects
    If igxDiagramObj.Type = ixObjectShape Then
        MsgBox "Conversion worked. Found a shape object."
    Else
        MsgBox "Conversion failed. Did not find a shape object."
    End If
Next igxDiagramObj
```

See Also Shape object

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

Detach Method

Syntax *TextGraphicObject*.**Detach**()

Description The Detach method breaks an existing attachment between a TextGraphicObject and any other object. This method provides the opposite function of the Attach method.

Example The following example attaches a TextGraphicObject to a shape and turns on a blue dashed callout line. Then it detaches the TextGraphicObject from the shape.

```
' Dimension the variables.
Dim igxTextGraphicObject As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Create a text graphic object on the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects _
    .AddTextObject(1440, 1440 * 4, , , "A string of text")
' Set the properties of the callout line
Set igxLineFormat = igxTextGraphicObject.CalloutLineFormat
' Set the callout line to vbBlue
igxLineFormat.Color = vbBlue
' Set the callout line style to dashed
igxLineFormat.Style = ixLineDashed
' Select the shape
Set igxDiagramObject = ActiveDiagram.DiagramObjects.Item(1)
' Attach the callout line from the text graphic object to the shape
igxTextGraphicObject.AttachTo igxDiagramObject
MsgBox "Attached callout line to shape"
igxTextGraphicObject.Detach
MsgBox "Detached callout line from shape"
```

See Also AttachTo method

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

DiagramObject Property

Syntax *TextGraphicObject*.**DiagramObject**

Data Type DiagramObject object (read-only, See Object Properties)

Description The DiagramObject property returns the DiagramObject object associated with the specified TextGraphicObject. For more information, see the description of extenders in DiagramObject topic.

Example The following example creates a TextGraphicObject in the active diagram. Then, using the DiagramObject property to access the “DiagramObject-level”, it selects the TextGraphic and moves it to a new location. Then it displays the ObjectName and Name properties. Note that the ObjectName property is blank, so you can give the object an name if you want. The Name property always returns the string “DiagramObject”, and is a convention more than a useful property.

```
' Dimension the variables
Dim igxTGObj As TextGraphicObject
Dim igxDiagramObject As DiagramObject
' Create a shape in the active diagram
Set igxTGObj = ActiveDiagram.DiagramObjects.AddTextObject _
    (1440, 1440, 1440 * 2, 720, "I am a TextGraphic, without _
    a graphic")
MsgBox "Created a TextGraphic object"
' Get the DiagramObject object of the new shape
Set igxDiagramObject = igxTGObj.DiagramObject
' Select the TextGraphic
igxDiagramObject.Selected = True
MsgBox "TextGraphic has been selected"
' Move the TextGraphic
igxDiagramObject.CenterX = 1440 * 3
igxDiagramObject.CenterY = 1440 * 4
' Display the ObjectName and Name properties
MsgBox "Object name: " & igxDiagramObject.ObjectName _
    & Chr(13) & "Name: " & igxDiagramObject.Name
```

See Also DiagramObject object

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

FillFormat Property

Syntax *TextGraphicObject*.**FillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The FillFormat property returns the FillFormat object for the specified TextGraphicObject object. The FillFormat object controls whether a fill is used, and if so, what type of fill (solid, pattern, or gradient), and the color or colors used.

Note that a Graphic object associated with the TextGraphicObject may use the ProtectFillFormat property to protect its fill formatting, in which case any fill formatting specified for the TextGraphicObject has no affect on that graphic.

Example The following example creates a TextGraphicObject and gives it a solid yellow fill.

```
' Dimension the variables
Dim igxTextGraphicObject As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxFillFormat As FillFormat
' Create a text graphic object on the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440, , , "A string of text")
Set igxFillFormat = igxTextGraphicObject.FillFormat
igxFillFormat.FillType = ixFillSolid
igxFillFormat.FillColor = vbYellow
MsgBox "View the diagram"
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

Graphic Property

Syntax *TextGraphicObject*.**Graphic**

Data Type Graphic object (read-only, See Object Properties)

Description The Graphic property returns the Graphic object for the specified TextGraphicObject object. The Graphic object provides access to the graphical part of the TextGraphicObject, allowing you to set or query its properties, or invoke its methods.

Example The following example creates a shape and a TextGraphic in the active diagram. When created, the TextGraphic does not have a graphical part—just text. Using the Replace method, the TextGraphic is given the same graphic as the shape. The graphic's type is then displayed, and based on the type, some modifications are made to the TextGraphicObject.

```
' Dimension the variables
Dim igxTGObj As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxGraphicBuilder As New GraphicBuilder
Dim igxDiagramObject As DiagramObject
Dim igxShadowFormat As ShadowFormat
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a text graphic object on the active diagram
Set igxTGObj = ActiveDiagram.DiagramObjects.AddTextObject _
   (1440 * 2, 1440 * 2, , , "Text Graphic")
MsgBox "Created a shape, and a TextGraphic object with no graphic"
' Set the graphic of the TextGraphic object to the same graphic
' used by the shape
igxTGObj.Graphic.Replace igxShape.Graphic
MsgBox "TextGraphic object given the shape's graphic"
' Display the type of the graphic
MsgBox "The graphic primitive type for the TextGraphic object " _
   & "is: " & igxTGObj.Graphic.Type
' Based on the type of graphic primitive, change some properties
' Ignore cases of images and groups
Select Case igxTGObj.Graphic.Type
   Case ixGraphicEllipse:
      igxTGObj.Graphic.EllipseGraphic.Width = 1440 * 1.5
      igxTGObj.Graphic.EllipseGraphic.Height = 1440
      igxTGObj.Graphic.EllipseGraphic.Left = 1440
      MsgBox "Modified the ellipse graphic"
   Case ixGraphicPolygon:
      MsgBox "Graphic type is a polygon. Adjusting the size through" _
          & Chr(13) & "the DiagramObject level."
      igxTGObj.DiagramObject.Height = 1440
      igxTGObj.DiagramObject.Width = 1440 * 2
      MsgBox "Modified the polygon graphic"
   Case ixGraphicPolyPolygon:
      MsgBox "Graphic type is a polypolygon. It consists of " _
          & igxTGObj.Graphic.PolyPolygonGraphic.Count & " polygons."
   Case ixGraphicRectangle:
      igxTGObj.Graphic.RectangleGraphic.Width = 1440 * 1.5
      igxTGObj.Graphic.RectangleGraphic.Height = 1440
```

```
        igxTGObj.Graphic.RectangleGraphic.Left = 1440
        MsgBox "Modified the rectangle graphic"
    Case ixGraphicArc:
        igxTGObj.Graphic.ArcGraphic.Left = 1440
        igxTGObj.Graphic.ArcGraphic.Top = 1440 * 4
        MsgBox "Moved the arc graphic"
End Select
```

See Also Graphic object

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

IsAttached Property

Syntax *TextGraphicObject*.**IsAttached** [= {**True** | **False**}]

Data Type Boolean (read-only)

Description The IsAttached property indicates whether the specified TextGraphicObject is attached to anything, such as a shape or a connector line.

Example The following example creates a shape and a TextGraphicObject in the active diagram. It then attaches the TextGraphicObject to the shape, and tests whether the attach operation worked by using the IsAttached property. It displays the appropriate message based on the IsAttached property's value.

```
' Dimension the variables
Dim igxTGObj As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
Dim igxShadowFormat As ShadowFormat
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape.Text = "Shape 1"
' Create a text graphic object on the active diagram
Set igxTGObj = ActiveDiagram.DiagramObjects.AddTextObject _
    (1440 * 2, 1440 * 2, , , "Text Graphic")
MsgBox "View the diagram"
' Set the properties of the callout line
Set igxLineFormat = igxTGObj.CalloutLineFormat
igxLineFormat.Color = vbBlue
igxLineFormat.Style = ixLineDashed
igxLineFormat.Width = 40
' Select the shape on the active diagram
Set igxDiagramObject = ActiveDiagram.DiagramObjects.Item(1)
' Attach the callout line from the text graphic object to the shape
igxTGObj.AttachTo igxDiagramObject
MsgBox "View the diagram"
' Test whether the text graphic is attached, and display the name of
' the object it is attached to
If (igxTGObj.IsAttached) Then
    MsgBox "TextGraphicObject is attached." & Chr(13) _
        & "It is attached to " & igxShape.Text
Else
    MsgBox "TextGraphicObject is NOT attached to anything."
End If
```

See Also AttachTo method

Detach method

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

LineFormat Property

Syntax *TextGraphicObject*.**LineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The LineFormat property returns the LineFormat object for the specified TextGraphicObject object. This property allows you to change all of the line formatting attributes of the TextGraphicObject, such as color, style, and width.

Note that a Graphic object associated with the TextGraphicObject may use the ProtectLineFormat property to protect its line formatting, in which case any line formatting specified for the TextGraphicObject has no affect on that graphic.

Example The following example creates a shape and a TextGraphicObject, with text only, in the active diagram. It then creates an ellipse with the GraphicBuilder, and sets the height and width, fill and line format of the ellipse. Next, the current graphic of the TextGraphicObject is replaced with the graphic from the shape, and the TextGraphic is resized. Then, the graphic is again replaced, this time with the ellipse from the GraphicBuilder.. Then the LineFormat property is used to set the graphic's line characteristics to a red, dashed, 2 point line.

```
' Dimension the variables
Dim igxTGObj As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxGraphicBuilder As New GraphicBuilder
Dim igxDiagramObject As DiagramObject
' Create the shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a text graphic object in the active diagram
Set igxTGObj = ActiveDiagram.DiagramObjects.AddTextObject _
    (1440 * 2, 1440 * 2, , , "Text Graphic")
MsgBox "Created a shape and a TextGraphic"
' Create an ellipse with the GraphicBuilder
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
igxGraphicBuilder.Graphic.EllipseGraphic.Height = 1440 - 360
igxGraphicBuilder.Graphic.EllipseGraphic.Width = 1440
igxGraphicBuilder.Graphic.FillFormat.FillType = ixFillNone
igxGraphicBuilder.Graphic.LineFormat.Style = ixLineNormal
' Replace the graphic in the TextGraphic with the new graphic
MsgBox "Click OK to replace the graphic."
igxTGObj.Graphic.Replace igxShape.Graphic
MsgBox "Click OK to continue."
' Resize the TextGraphic
igxTGObj.DiagramObject.Height = 1440
igxTGObj.DiagramObject.Width = 1440
MsgBox "TextGraphic resized"
' Now replace the TextGraphic's graphic with the GraphicBuilder graphic
igxTGObj.Graphic.Replace igxGraphicBuilder.Graphic
MsgBox "TextGraphic's graphic changed to an ellipse"
' Set the line format properties
Set igxLineFormat = igxTGObj.LineFormat
' Set the line color to red, line style to dashed, and width
' to 2 points
igxLineFormat.Color = vbRed
igxLineFormat.Style = ixLineDashed
```

```
igxLineFormat.Width = 40
MsgBox "Line format of TextGraphic changed"
```

See Also

LineFormat object

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

Paragraphs Property

Syntax *TextGraphicObject*.**Paragraphs**

Data Type Paragraphs collection object (read-only, Object Properties)

Description The Paragraphs property returns the Paragraphs collection associated with the specified TextGraphicObject object. The Paragraphs object, through the Item method, provides access to the individual Paragragh objects.

Example The following example creates a shape and a TextGraphic object in the active diagram. It gives the text graphic four paragraphs, and then uses the Paragraphs property to access the last three paragraphs to set their horizontal alignment.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTGObj As TextGraphicObject
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Create a text graphic object on the active diagram
Set igxTGObj = ActiveDiagram.DiagramObjects.AddTextObject _
    (1440 * 2, 1440 * 2, , , "Text Graphic")
MsgBox "View the diagram"
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Make the TextGraphicObject wider
igxTGObj.DiagramObject.Width = 1440 * 3
' Add several new paragraphs to the text graphic
igxTGObj.Text = igxTGObj.Text & Chr(13) & "Paragraph 1" _
    & Chr(13) & "Paragraph 2" & Chr(13) & "Paragraph 3"
MsgBox "View the diagram. There are " _
    & igxTGObj.Paragraphs.Count _
    & " paragraphs in the text graphic object."
' Set a different bullet type for each paragraph
igxTGObj.Paragraphs.Item(2).ParagraphFormat.Alignment _
    = ixHorizontalAlignLeft
MsgBox "View the diagram"
igxTGObj.Paragraphs.Item(3).ParagraphFormat.Alignment _
    = ixHorizontalAlignRight
MsgBox "View the diagram"
igxTGObj.Paragraphs.Item(4).ParagraphFormat.Alignment _
    = ixHorizontalAlignCenter
MsgBox "View the diagram"
```

See Also Paragraph object

Paragraphs object

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

ShadowFormat Property

Syntax *TextGraphicObject*.**ShadowFormat**

Data Type ShadowFormat object (read-only, See Object Properties)

Description The ShadowFormat property returns the ShadowFormat object for the specified TextGraphicObject object.

Example The following example draws a shape and a TextGraphicObject in the diagram. It attaches a callout line from the TextGraphicObject to the shape, and then gives the TextGraphicObject a red shadow, with a depth of 3.

```
' Dimension the variables
Dim igxTGObj As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
Dim igxShadowFormat As ShadowFormat
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a text graphic object on the active diagram
Set igxTGObj = ActiveDiagram.DiagramObjects.AddTextObject _
    (1440 * 2, 1440 * 2, , , "Text")
MsgBox "View the diagram"
' Set the properties of the callout line
Set igxLineFormat = igxTGObj.CalloutLineFormat
' Set the callout line to vbBlue
igxLineFormat.Color = vbBlue
' Set the callout line style to dashed
igxLineFormat.Style = ixLineDashed
' Set the width of the line to 2 points
igxLineFormat.Width = 40
' Select the shape on the active diagram
Set igxDiagramObject = ActiveDiagram.DiagramObjects.Item(1)
' Attach the callout line from the text graphic object to the shape
igxTGObj.AttachTo igxDiagramObject
MsgBox "View the diagram"
Set igxShadowFormat = igxTGObj.ShadowFormat
igxShadowFormat.Color = vbRed
igxShadowFormat.Depth = 3
igxShadowFormat.Type = ixShadow14
MsgBox "View the diagram"
```

See Also ShadowFormat object

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

TextRange Property

Syntax *TextGraphicObject*.**TextRange**([*First* As Long = 1], [*Last* As Long])

Data Type TextRange object (read-only, See Object Properties)

Description The TextRange property returns a TextRange object for the specified TextGraphicObject object. The purpose of this property is to provide control over a range of text within a TextGraphicObject.

The TextRange object lets you work with a range of text. The *First* and *Last* arguments specify the start and end positions of the text range. For example, specifying Paragraph1.TextRange(1,5) returns a TextRange that contains the first five characters of the paragraph. Specifying the property without providing the *First* and *Last* arguments returns a TextRange with all the characters in the paragraph. The *First* argument defaults to a value of 1, so to select from the first character of the paragraph only requires specifying the last character.

In addition, each Paragraph object contained within a TextGraphicObject has its own TextRange object that can be used to select either all or part of the paragraph.

Example The following example creates a TextGraphicObject in the diagram with a text string. Then part of the first paragraph is extracted into a text range. The text range is then used in the specification of a second paragraph.

```
' Dimension the variables
Dim igxTextGraphicObj As TextGraphicObject
Dim igxParagraph As Paragraph
Dim igxTextRange As TextRange
' Create a TextGraphic in the active diagram
Set igxTextGraphicObj = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440, , , "A string of text")
' Set igxParagraph variable to the first Paragraph object
Set igxParagraph = igxTextGraphicObj.Paragraphs.Item(1)
' Get the TextRange object
Set igxTextRange = igxParagraph.TextRange(3, 8)
' Display the text within the TextRange object
MsgBox "The text in the TextRange object is '" _
    & igxTextRange.Text & "'"
' Create a second paragraph, inserting the TextRange text
igxParagraph.TextLF = igxParagraph.Text & Chr$(13) _
    & "New text with a " & igxTextRange.Text
MsgBox "There are now " & igxTextGraphicObj.Paragraphs.Count _
    & " paragraphs."
```

See Also TextRange object

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

ThreeDFormat Property

Syntax *TextGraphicObject*.**ThreeDFormat**

Data Type ThreeDFormat object (read-only, See Object Properties)

Description The ThreeDFormat property returns the ThreeDFormat object for the specified TextGraphicObject object. Use the object to give the TextGraphicObject a 3D effect.

Example The following example draws a shape and a TextGraphicObject in the diagram. It attaches a callout line from the TextGraphicObject to the shape, and then gives the TextGraphicObject a 3D effect.

```
' Dimension the variables
Dim igxTGObj As TextGraphicObject
Dim igxLineFormat As LineFormat
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
Dim igxThreeDFormat As ThreeDFormat
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a text graphic object on the active diagram
Set igxTGObj = ActiveDiagram.DiagramObjects.AddTextObject _
    (1440 * 2, 1440 * 2, , , "Text")
MsgBox "View the diagram"
' Set the properties of the callout line
Set igxLineFormat = igxTGObj.CalloutLineFormat
' Set the callout line to vbBlue
igxLineFormat.Color = vbBlue
' Set the callout line style to dashed
igxLineFormat.Style = ixLineDashed
' Set the width of the line to 2 points
igxLineFormat.Width = 40
' Select the shape on the active diagram
Set igxDiagramObject = ActiveDiagram.DiagramObjects.Item(1)
' Attach the callout line from the text graphic object to the shape
igxTGObj.AttachTo igxDiagramObject
MsgBox "View the diagram"
Set igxThreeDFormat = igxTGObj.ThreeDFormat
igxThreeDFormat.Type = ixThreeD4
MsgBox "View the diagram"
```

See Also ThreeDFormat object

iGrafx API Object Hierarchy

{button TextGraphicObject object,JI(`igrafxrf.HLP',`TextGraphicObject_Object')}

TextRange Object

The TextRange object allows you to work with a range of text, providing access to text on a character level. The same types of actions that can be applied to a selected range of text through the user interface are available to the programmer through this object (although the text range you are working with is not shown as "selected" in the user interface.)

You can use the TextRange object to change the font, font style, and font size of a range of characters, or insert field codes into a text range, or cut, copy and paste a text range.

To access the actual text of a TextRange, you can use either the Text or the TextLF property. Both properties return the entire text string for the TextRange. The difference is that the TextLF property preserves any carriage returns in the string, while the Text property replaces carriage returns with spaces.

Properties, Methods, and Events

All of the properties, methods, and events for the TextRange object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Copy	
FieldTexts	Cut	
Font	InsertFieldText	
Parent	Paste	
Text		
TextLF		

Related Topics

ChildTextBlock object
Department object
FieldTexts object
HeaderFooter object
Note object
Paragraph object
TextBlock object
TextGraphicObject object

Copy Method

Syntax *TextRange*.**Copy**

Description The Copy method copies the selected text range and places it onto the Windows clipboard. Once copied onto the clipboard, the text can be pasted to some other location using the Paste method.

Example The following example gets a text range from a text graphic object and copies and pastes it into a new shape.

```
' Dimension the variables.
Dim igxShape As Shape
Dim igxTextGraphicObject As TextGraphicObject
Dim igxTextRange As TextRange
' Create a text graphic object in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects _
    .AddTextObject(4000, 4000, , , "A string of text")
' Get the TextRange object
Set igxTextRange = igxTextGraphicObject.TextRange(3, 9)
' Display the text within the TextRange object
MsgBox "The text in the TextRange object is " _
    & igxTextRange.Text
igxTextRange.Copy
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.TextBlock.TextRange.Paste
MsgBox "View the shape text"
```

See Also Cut method

Paste method

{button TextRange object,JI(`igrafxrf.HLP',`TextRange_Object')}

Cut Method

Syntax *TextRange*.**Cut**

Description The Cut method cuts the selected text range and places it onto the Windows clipboard. Once placed onto the clipboard, the text can be pasted to some other location using the Paste method.

Example The following example gets a text range from a text graphic object and cuts and pastes it into a new shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextGraphicObject As TextGraphicObject
Dim igxTextRange As TextRange
' Create a text graphic object in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects _
    .AddTextObject(4000, 4000, , , "A string of text")
' Get the TextRange object
Set igxTextRange = igxTextGraphicObject.TextRange(3, 9)
' Display the text within the TextRange object
MsgBox "The text in the TextRange object is " _
    & igxTextRange.Text
igxTextRange.Cut
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.TextBlock.TextRange.Paste
MsgBox "View the shape text"
```

See Also Copy method

Paste method

{button TextRange object,JI(`igrafxrf.HLP',`TextRange_Object')}

FieldTexts Property

Syntax *TextRange*.**FieldTexts**

Data Type FieldTexts collection object (read-only, See Object Properties)

Description The FieldTexts property returns the FieldTexts collection associated with the specified TextRange object. The FieldTexts collection contains any field codes used in a range of text.

A TextRange could, potentially, contain any number of FieldText objects, such as the current date, the diagram name, etc., that you may want to gain access to. This property provides that access. You use the InsertFieldText method to add a FieldText object into the text range.

Example The following example creates a TextGraphicObject in the active diagram that has some initial text, “Date Created: “. This was set up so that a FieldText object containing the creation date could be added to the end of the string. First, the initial string is captured in a TextRange. Then, using the TextRange.InsertFieldText method, the creation date is added to the text range, with an initial format type of Year/Month/Day. Then the FieldTexts property is used to access the FieldText object in the text range, and change the date formatting to Day/Month/Year.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextGraphicObject As TextGraphicObject
Dim igxTextRange As TextRange
Dim igxFieldText As FieldText
' Create a text graphic object in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects _
    .AddTextObject(1440, 1440, , , "Date Created: ")
' Get the TextRange object; select the entire string
Set igxTextRange = igxTextGraphicObject.TextRange()
' Display the text within the TextRange object
MsgBox "The text in the TextRange object is " & igxTextRange.Text
Set igxFieldText = igxTextRange.InsertFieldText _
    (15, ixFieldTextCreateDate, ixDateTextYearMonthDay)
MsgBox "The Creation date was added to the end of the Text range"
' Check the Count in the FieldTexts collection
MsgBox "The TextRange FieldTexts collection contains " _
    & igxTextRange.FieldTexts.Count & " item."
' Change the format of the creation date from Year/Month/Day
' to Day/Month/Year
igxTextRange.FieldTexts.Item(1).DateFormat = ixDateTextDayMonthYear
MsgBox "View the diagram"
```

See Also InsertFieldText method

FieldTexts object

iGrafx API Object Hierarchy

{button TextRange object,JI(`igrafxrf.HLP',`TextRange_Object')}

Font Property

Syntax *TextRange*.**Font**

Data Type Font object (read-only, See Object Properties)

Description The Font property returns the Font object associated with the specified TextGraphicObject object. You use this property to change the font, font style, font size, and font color of the text range.

Example The following example gets a text range from a text graphic object and copies and pastes it into a new shape. It then changes the font color to green.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextGraphicObject As TextGraphicObject
Dim igxTextRange As TextRange
' Create a text graphic object in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects _
    .AddTextObject(4000, 4000, , , "A string of text")
' Get the TextRange object
Set igxTextRange = igxTextGraphicObject.TextRange(3, 9)
' Display the text within the TextRange object
MsgBox "The text in the TextRange object is " & igxTextRange.Text
igxTextRange.Copy
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.TextBlock.TextRange.Paste
MsgBox "View the shape text"
igxShape.TextBlock.TextRange.Font.Color = vbGreen
igxShape.TextBlock.TextRange.Font.Bold = True
MsgBox "View the shape text"
```

See Also Font object

iGrafx API Object Hierarchy

{button TextRange object,JI(`igrafxrf.HLP',`TextRange_Object')}

InsertFieldText Method

Syntax *TextRange*.**InsertFieldText**(*InsertPosition* As Long, *FieldTextType* As IxFieldTextType, *Value* As Variant) As FieldText

Description The InsertFieldText method inserts one of the available field codes (see the FieldCodes property) into a selected text range. The result of this method must be assigned to a variable of type “FieldText”. The method’s arguments provide the following data:

- The *InsertPosition* argument specifies the insertion position within the range of characters of the TextRange object.
- The *FieldTextType* argument specifies the type of field code to insert.
- The *Value* argument specifies the actual value that the FieldText object contains. The value you specify depends on which FieldTextType is used.

The IxFieldTextType constant defines the valid values for this property, which are listed in the following table. Information about the *Value* argument is also given in the table

Value	Name of Constant	Description of the Value Argument
-1	ixFieldCodeNone	Unused; specify empty double quotes (“”).
0	ixFieldTextPageNumber	Unused; specify empty double quotes (“”).
1	ixFieldTextPageCount	Unused; specify empty double quotes (“”).
2	ixFieldTextDiagramName	Unused; specify empty double quotes (“”).
3	ixFieldTextFileName	Unused; specify empty double quotes (“”).
4	ixFieldTextCurrentDate	*Value* should be a numeric variant. It controls the time/date format displayed. Use the values from the IxDateFormatType constant.
5	ixFieldTextCreateDate	*Value* should be a numeric variant. It controls the time/date format displayed. Use the values from the IxDateFormatType constant.
6	ixFieldTextSaveDate	*Value* should be a numeric variant. It controls the time/date format displayed. Use the values from the IxDateFormatType constant.
7	ixFieldTextDataField	*Value* can be a string, specifying the custom data item's name, or a number, giving its ID.
10	ixFieldTextExpression	*Value* is a string specifying a Visual Basic expression.
12	ixFieldTextShapeNote	Unused; specify empty double quotes (“”).
13	ixFieldTextShapeNumber	Unused; specify empty double quotes (“”).

Example The following example creates a TextGraphicObject in the active diagram that has some initial text, “Diagram Name: “. This was set up so that a FieldText object containing the name of the diagram could be added to the end of the string. First, the initial string is captured in a

TextRange. Then, using the TextRange.InsertFieldText method, the name of the diagram is added to the text range.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextGraphicObject As TextGraphicObject
Dim igxTextRange As TextRange
Dim igxFieldText As FieldText
' Create a text graphic object in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects _
    .AddTextObject(1440, 1440, , , "Diagram Name: ")
' Get the TextRange object; select the entire string
Set igxTextRange = igxTextGraphicObject.TextRange()
' Display the text within the TextRange object
MsgBox "The text in the TextRange object is " & igxTextRange.Text
Set igxFieldText = igxTextRange.InsertFieldText _
    (15, ixFieldTextDiagramName, "")
MsgBox "The Diagram Name was added to the end of the Text range"
```

See Also

FieldTexts property

FieldText object

{button TextRange object,JI(`igrafxrf.HLP',`TextRange_Object')}

Paste Method

Syntax *TextRange*.**Paste**

Description The Paste method pastes the contents of the Windows clipboard into the selected text range. The text to be pasted can be placed onto the Windows clipboard by using the Cut or Copy methods.

Example The following example gets a text range from a text graphic object and copies and pastes it into a new shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextGraphicObject As TextGraphicObject
Dim igxTextRange As TextRange
' Create a text graphic object in the active diagram
Set igxTextGraphicObject = ActiveDiagram.DiagramObjects _
    .AddTextObject(4000, 4000, , , "A string of text")
' Get the TextRange object
Set igxTextRange = igxTextGraphicObject.TextRange(3, 9)
' Display the text within the TextRange object
MsgBox "The text in the TextRange object is " & igxTextRange.Text
igxTextRange.Copy
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.TextBlock.TextRange.Paste
MsgBox "View the shape text"
```

See Also Cut method

Copy method

{button TextRange object,JI(`igrafxrf.HLP',`TextRange_Object')}

ChildTextBlock Object

The ChildTextBlock object, like the TextBlock object, is a container for text. Every Shape and ShapeClass object has a TextBlock object associated with it. Additional text areas can be created within the TextBlock object using the ChildTextBlocks collection and ChildTextBlock object.

A ChildTextBlock takes space away from the main TextBlock object for a shape or shape class; that is, child text block areas are created inside the area of the main text block.

Text formatting within a ChildTextBlock object is controlled by the BlockFormat object, and the Paragraph object (accessed through the Paragraphs collection).

Properties, Methods, and Events

All of the properties, methods, and events for the ChildTextBlock object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
BlockFormat		
Paragraphs		
Parent		
Position		
SizeFixed		
SizePercentage		
SizeType		
Text		
TextLF		
TextMargin		
TextRange		

Related Topics

BlockFormat object

ChildTextBlocks object

TextBlock object

iGrafx API Object Hierarchy

BlockFormat Property

Syntax *ChildTextBlock*.**BlockFormat**

Data Type BlockFormat object (read-only, See Object Properties)

Description The BlockFormat property returns the BlockFormat object associated with the specified ChildTextBlock object.

The BlockFormat object controls the formatting of the text associated with a shape (the TextBlock or ChildTextBlock objects). The TextBlock object (there is only one per shape) and all ChildTextBlock objects (there can be zero or more per TextBlock) have there own distinct BlockFormat objects for controlling text formatting.

Example The following example creates a shape in the diagram with a child text block positioned at the top, and with four paragraphs of text. It then uses the BlockFormat object to make the child text block's border a dashed red line, set its fill color to green, align the text to the bottom of the block, and make the text opaque.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 3
igxShape.DiagramObject.Width = 1440 * 3
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Add a child text block at the top, sized at 40%
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextTop, 40.0)
' Add text to the child text block
igxChildTextBlk.Text = "Child Text Block" & Chr(13) _
    & "Paragraph 1" & Chr(13) & "Paragraph 2" & Chr(13) _
    & "Paragraph 3"
MsgBox "View the diagram"
' Set various properties of the child text block through
' the BlockFormat object
With igxChildTextBlk.BlockFormat.LineFormat
    .Color = vbRed
    .Style = ixLineDashed
    .Width = 1
End With
MsgBox "View the diagram"
' Give the child text block a green fill
igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbGreen
MsgBox "View the diagram"
' Align the text to the bottom
```

```
igxChildTextBlk.BlockFormat.VerticalAlignment = ixVerticalAlignBottom
MsgBox "View the diagram"
igxChildTextBlk.BlockFormat.Opaque = True
MsgBox "View the diagram"
```

See Also

BlockFormat object

iGrafx API Object Hierarchy

{button ChildTextBlock object,JI(`igrafxrf.HLP',`ChildTextBlock_Object')}

Paragraphs Property

Syntax *ChildTextBlock*.**Paragraphs**

Data Type Paragraphs collection object (read-only, See Object Properties)

Description The Paragraphs property returns the Paragraphs collection associated with the specified ChildTextBlock object. The Paragraphs object, through the Item method, provides access to the individual Paragragh objects.

Example The following example creates a shape with a child text block positioned at the top. It places four paragraphs of text in the child text block, and then uses the Paragraphs object to access then second, third, and fourth paragraphs to change their horizontal alignment.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 3
igxShape.DiagramObject.Width = 1440 * 3
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Add a child text block at the top, sized at 40%
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextTop, 40.0)
With igxChildTextBlk.BlockFormat.LineFormat
    .Color = vbRed
    .Style = ixLineDashed
    .Width = 1
End With
' Give the child text block a yellow fill
igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbYellow
MsgBox "View the diagram"
' Add text to the child text block
igxChildTextBlk.Text = "Child Text Block" & Chr(13) _
    & "Paragraph 1" & Chr(13) & "Paragraph 2" & Chr(13) _
    & "Paragraph 3"
MsgBox "View the diagram. There are " _
    & igxChildTextBlk.Paragraphs.Count _
    & " paragraphs in the child text block."
' Set a different bullet type for each paragraph
igxChildTextBlk.Paragraphs.Item(2).ParagraphFormat.Alignment _
    = ixHorizontalAlignLeft
MsgBox "View the diagram"
igxChildTextBlk.Paragraphs.Item(3).ParagraphFormat.Alignment _
    = ixHorizontalAlignRight
```

```
MsgBox "View the diagram"
igxChildTextBlk.Paragraphs.Item(4).ParagraphFormat.Alignment _
    = ixHorizontalAlignCenter
MsgBox "View the diagram"
```

See Also

Paragraph object

Paragraphs object

iGrafx API Object Hierarchy

{button ChildTextBlock object,JI(`igrafxrf.HLP',`ChildTextBlock_Object')}

Position Property

Syntax *ChildTextBlock*.**Position**

Data Type IxChildTextPosition enumerated constant (read/write)

Description The Position property specifies the location at which to place a ChildTextBlock object within the main TextBlock of a shape. Note that you are not limited to adding only one child text block at each position; for instance, it is just as valid to specify two child text blocks positioned at the top as it is to position one at the top and one at the bottom.

The size of the ChildTextBlock is controlled by the Size and SizeType properties.

The IxChildTextPosition constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant
0	ixTextLeft
1	ixTextTop
2	ixTextRight
3	ixTextBottom

Example The following example creates a shape in the diagram and gives it a green border. Then, four child text blocks are added to the shape, one at each of the four possible positions. The left and right ones are sized at 20%, and the top and bottom ones at 30%. Finally, a fifth child text block is added, again on the right, sized at 20% and given a blue fill.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 3
igxShape.DiagramObject.Width = 1440 * 3
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Add two child text blocks, one left and one right, both sized at 20%
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextLeft, 20.0)
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextRight, 20.0)
MsgBox "View the diagram"
' Add two child text blocks, one top and one bottom, both sized at 30%
Set igxChildTextBlk = _
```

```
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextTop, 30.0)
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextBottom, 30.0)
MsgBox "View the diagram"
For iCount = 1 To igxShape.TextBlock.ChildTextBlocks.Count
   Set igxChildTextBlk = _
        igxShape.TextBlock.ChildTextBlocks.Item(iCount)
   With igxChildTextBlk.BlockFormat.LineFormat
       .Color = vbRed
       .Style = ixLineDashed
       .Width = 1
   End With
   ' Add text to the child text block
   igxChildTextBlk.Text = "Child Text Block " & iCount
   ' Give the child text block a yellow fill
   igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbYellow
   MsgBox "View the diagram"
Next iCount
' Add one last text block, again positioning it at the right
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextRight, 20.0)
With igxChildTextBlk.BlockFormat.LineFormat
       .Color = vbRed
       .Style = ixLineDashed
       .Width = 1
End With
igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbBlue
MsgBox "View the diagram"
```

See Also

SizeFixed property

SizePercentage property

SizeType property

{button ChildTextBlock object,JI(`igrafxrf.HLP',`ChildTextBlock_Object')}

SizeFixed Property

Syntax *ChildTextBlock*.**SizeFixed**

Data Type Long (read/write)

Description The SizeFixed property specifies the size of the specified ChildTextBlock object, in fixed units of twips (1440 twips = 1 inch). A child text block is part of a shape's main text block; therefore, it takes space away from the main text block according to the size specification. This property is used only if the SizeType property is set to ixTextPositionFixed. The value of this property is set when a child text block is added using the AddFixed method.

If the Position property is left or right, the SizeFixed property represents the width; if the Position property is top or bottom, the property represents the height.

Example The following example creates a shape, and sets border line colors for both the shape and the main text block. Two child text blocks are added, one on the left and one on the right, using the AddPercentage to make them each 30% of the shape's size. Then two more child text blocks are added, each specified at 10%. The shape is then made wider to show that the child text blocks still take up the same percentage of the shape's overall size. Then the size of the child text blocks are changed using the appropriate "Size" property. Next, a second shape is added, set up the same way as the first, but then two child text blocks are added using the AddFixed method. When the shape is resized, the child text blocks stay at their originally specified, fixed size. Finally, these two child text blocks sizes are changed with the appropriate "Size" property.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeDup As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 1.5
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineDashed
    .Width = 1
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "Dimensions are: " _
   & igxShape.DiagramObject.Width _
   & " X " & igxShape.DiagramObject.Height
MsgBox "The main text block is the same size as the shape"
' Add two child text blocks, one left and one right, both sized at 30%
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
```

```
    (ixTextLeft, 30.0)
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextRight, 30.0)
MsgBox "View the diagram"
' Add two child text blocks, one left and one right, both sized at 10%
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextLeft, 10.0)
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextRight, 10.0)
MsgBox "View the diagram"
' Make the shape wider
igxShape.DiagramObject.Width = 1440 * 4
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
igxShape.TextBlock.Text = "Dimensions are: " & _
   igxShape.DiagramObject.Width _
   & " X " & igxShape.DiagramObject.Height
MsgBox "Change the sizes of the child text blocks"
For Each igxChildTextBlk In igxShape.TextBlock.ChildTextBlocks
    If (igxChildTextBlk.SizeType = ixTextPositionFixed) Then
        If (igxChildTextBlk.SizeFixed < 360) Then
            igxChildTextBlk.SizeFixed = 540
        Else
            igxChildTextBlk.SizeFixed = 270
        End If
    Else
        If (igxChildTextBlk.SizePercentage < 25.0) Then
            igxChildTextBlk.SizePercentage = 35.0
        Else
            igxChildTextBlk.SizePercentage = 10.0
        End If
    End If
   MsgBox "View the change"
Next igxChildTextBlk
' Make a second shape and position it at X = 1, Y = 3
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 1.5
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Top = 1440 * 3
igxShape.DiagramObject.Left = 1440
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineDashed
    .Width = 1
End With
```

```
' Add text to the Text Block
igxShape.TextBlock.Text = "Dimensions are: " _
    & igxShape.DiagramObject.Width _
    & " X " & igxShape.DiagramObject.Height
MsgBox "View the diagram"
' Add two child text blocks, one left, the other right, both sized
' at 720 twips, or 1/2 inch
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddFixed _
    (ixTextLeft, 720)
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddFixed _
    (ixTextRight, 720)
MsgBox "View the diagram"
' Make the shape wider
igxShape.DiagramObject.Width = 1440 * 4
igxShape.DiagramObject.Top = 1440 * 3
igxShape.DiagramObject.Left = 1440
igxShape.TextBlock.Text = "Dimensions are: " & _
   igxShape.DiagramObject.Width _
   & " X " & igxShape.DiagramObject.Height
MsgBox "Change the sizes of the child text blocks"
For Each igxChildTextBlk In igxShape.TextBlock.ChildTextBlocks
    If (igxChildTextBlk.SizeType = ixTextPositionFixed) Then
        If (igxChildTextBlk.SizeFixed < 360) Then
            igxChildTextBlk.SizeFixed = 540
        Else
            igxChildTextBlk.SizeFixed = 270
        End If
    Else
        If (igxChildTextBlk.SizePercentage < 25.0) Then
            igxChildTextBlk.SizePercentage = 35.0
        Else
            igxChildTextBlk.SizePercentage = 10.0
        End If
    End If
   MsgBox "View the change"
Next igxChildTextBlk
MsgBox "End of example"
```

See Also

Position property

SizePercentage property

SizeType property

{button ChildTextBlock object,JI(`igrafxrf.HLP',`ChildTextBlock_Object')}

SizePercentage Property

Syntax *ChildTextBlock*.**SizePercentage**

Data Type Double (read/write)

Description The SizePercentage property specifies the size of the specified ChildTextBlock object, as a percentage of the shape's size; for example, 10%, 25%, etc. Valid values range from greater than 0.0 to 100.0 or greater (if you want to specify a size of more than 100% of the shape's size). A child text block is part of a shape's main text block; therefore, it takes space away from the main text block according to the size specification. This property is used only if the SizeType property is set to ixTextPositionPercentage. The value of this property is set when a child text block is added using the AddPercentage method.

If the Position property is left or right, the SizePercentage property represents the width; if the Position property is top or bottom, the property represents the height.

Example The following example creates a shape, and sets border line colors for both the shape and the main text block. Two child text blocks are added, one on the left and one on the right, using the AddPercentage to make them each 30% of the shape's size. Then two more child text blocks are added, each specified at 10%. The shape is then made wider to show that the child text blocks still take up the same percentage of the shape's overall size. Then the size of the child text blocks are changed using the appropriate "Size" property. Next, a second shape is added, set up the same way as the first, but then two child text blocks are added using the AddFixed method. When the shape is resized, the child text blocks stay at their originally specified, fixed size. Finally, these two child text blocks sizes are changed with the appropriate "Size" property.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeDup As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 1.5
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineDashed
    .Width = 1
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "Dimensions are: " _
   & igxShape.DiagramObject.Width _
   & " X " & igxShape.DiagramObject.Height
MsgBox "The main text block is the same size as the shape"
' Add two child text blocks, one left and one right, both sized at 30%
Set igxChildTextBlk = _
```

```
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextLeft, 30.0)
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextRight, 30.0)
MsgBox "View the diagram"
' Add two child text blocks, one left and one right, both sized at 10%
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextLeft, 10.0)
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextRight, 10.0)
MsgBox "View the diagram"
' Make the shape wider
igxShape.DiagramObject.Width = 1440 * 4
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
igxShape.TextBlock.Text = "Dimensions are: " & _
   igxShape.DiagramObject.Width _
   & " X " & igxShape.DiagramObject.Height
MsgBox "Change the sizes of the child text blocks"
For Each igxChildTextBlk In igxShape.TextBlock.ChildTextBlocks
    If (igxChildTextBlk.SizeType = ixTextPositionFixed) Then
        If (igxChildTextBlk.SizeFixed < 360) Then
            igxChildTextBlk.SizeFixed = 540
        Else
            igxChildTextBlk.SizeFixed = 270
        End If
    Else
        If (igxChildTextBlk.SizePercentage < 25.0) Then
            igxChildTextBlk.SizePercentage = 35.0
        Else
            igxChildTextBlk.SizePercentage = 10.0
        End If
    End If
   MsgBox "View the change"
Next igxChildTextBlk
' Make a second shape and position it at X = 1, Y = 3
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 1.5
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Top = 1440 * 3
igxShape.DiagramObject.Left = 1440
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineDashed
    .Width = 1
```

```
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "Dimensions are: " _
    & igxShape.DiagramObject.Width _
    & " X " & igxShape.DiagramObject.Height
MsgBox "View the diagram"
' Add two child text blocks, one left, the other right, both sized
' at 720 twips, or 1/2 inch
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddFixed _
    (ixTextLeft, 720)
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddFixed _
    (ixTextRight, 720)
MsgBox "View the diagram"
' Make the shape wider
igxShape.DiagramObject.Width = 1440 * 4
igxShape.DiagramObject.Top = 1440 * 3
igxShape.DiagramObject.Left = 1440
igxShape.TextBlock.Text = "Dimensions are: " & _
   igxShape.DiagramObject.Width _
   & " X " & igxShape.DiagramObject.Height
MsgBox "Change the sizes of the child text blocks"
For Each igxChildTextBlk In igxShape.TextBlock.ChildTextBlocks
    If (igxChildTextBlk.SizeType = ixTextPositionFixed) Then
        If (igxChildTextBlk.SizeFixed < 360) Then
            igxChildTextBlk.SizeFixed = 540
        Else
            igxChildTextBlk.SizeFixed = 270
        End If
    Else
        If (igxChildTextBlk.SizePercentage < 25.0) Then
            igxChildTextBlk.SizePercentage = 35.0
        Else
            igxChildTextBlk.SizePercentage = 10.0
        End If
    End If
   MsgBox "View the change"
Next igxChildTextBlk
MsgBox "End of example"
```

See Also

Position property

SizeFixed property

SizeType property

{button ChildTextBlock object,JI(`igrafxrf.HLP',`ChildTextBlock_Object')}

SizeType Property

Syntax *ChildTextBlock*.**SizeType**

Data Type IxTextPositionType enumerated constant (read/write)

Description The SizeType property specifies how a ChildTextBlock is sized. This property affects the value you specify for the Size property. If the SizeType is ixTextPositionFixed, then Size is specified in twips. If the SizeType is ixTextPositionPercentage, then Size is specified as a percentage of the size of the shape.

A SizeType of “Percentage” is often easier to use because you do not have to keep track of the actual size of the shape. For example, if you want to have two child text blocks the same size, one on the left and one on the right, then specifying the size as a percentage, say 25%, is easier than getting the width of the shape and dividing by four. For a sample of setting the size using each method, refer to the Example for the Size property.

The IxTextPositionType constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant
0	ixTextPositionFixed
1	ixTextPositionPercentage

Example The following example creates a shape and adds text to its text block. The border of the shape and the main text block are formatted with diffeent line styles and colors. The shape is then made larger, and two child text blocks are added, each using a different method of specifying the size in the ChildTextBlocks.Add method. Then both child text blocks are accessed from the collection, and both are set to use the same size type, resized, and the text is added along with line and fill formatting.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Set a border for the shape
igxShape.LineStyle = ixLineNormal
igxShape.LineColor = vbGreen
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineDashed
    .Width = 1
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "Main Text Block"
MsgBox "View the diagram"
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 3
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Top = 1440
```

```
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
' Add a ChildTextBlock at the left side of the main text block
' The size is in fixed units of twips; set to 1080, or 3/4 inch
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddFixed(ixTextLeft, 1080)
' Add a ChildTextBlock at the bottom of the remaining main text
' block, sized in relative units: 20% of the shape size
MsgBox "View the diagram"
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage _
    (ixTextBottom, 20.0)
MsgBox "View the diagram"
' Set the size type to "relative" for both child text blocks,
' resize them, and then set line format, fill format, and text
For iCount = 1 To igxShape.TextBlock.ChildTextBlocks.Count
   Set igxChildTextBlk = _
        igxShape.TextBlock.ChildTextBlocks.Item(iCount)
   igxChildTextBlk.SizeType = ixTextPositionPercentage
   igxChildTextBlk.SizePercentage = 25.0
   MsgBox "View the diagram"
   With igxChildTextBlk.BlockFormat.LineFormat
       .Color = vbRed
       .Style = ixLineDashed
       .Width = 1
   End With
   ' Add text to the child text block
   igxChildTextBlk.Text = "Child Text Block " & iCount
   ' Give the child text block a yellow fill
   igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbYellow
   MsgBox "View the diagram"
Next iCount
```

See Also

Position property

SizeFixed property

SizePercentage property

{button ChildTextBlock object,JI(`igrafxrf.HLP',`ChildTextBlock_Object')}

TextMargin Property

Syntax *ChildTextBlock*.**TextMargin**

Data Type Integer (read/write)

Description The TextMargin property specifies the size of the margins for the specified ChildTextBlock object. All margins from the child text block border (top, bottom, left, and right) are controlled by this property; that is, all margins are the same size in a child text block. You cannot set individual margins in a child text block like you can in the main text block. The units for this property are twips (1440 twips = 1 inch).

Example The following example creates a shape, sets up its text block, and then adds two child text blocks. It then uses the TextMargin property to set the margin for both child text blocks to 1/10 inch (144 twips).

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Set the shape's text block margins
With igxShape.TextBlock
    .BottomMargin = 0.1
    .LeftMargin = 0.1
    .RightMargin = 0.1
    .TopMargin = 0.1
End With
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineNormal
    .Width = 1
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "Main Text Block"
MsgBox "View the diagram"
' Add a ChildTextBlock at the left side of the main text block
' The size is in fixed units of twips; set to 360, or 1/4 inch
Set igxChildTextBlk = _
   igxShape.TextBlock.ChildTextBlocks.AddFixed(ixTextLeft, 360)
Set igxChildTextBlk = _
   igxShape.TextBlock.ChildTextBlocks.AddFixed(ixTextBottom, 360)
' Set the child text block's margin and line format
For iCount = 1 To igxShape.TextBlock.ChildTextBlocks.Count
   Set igxChildTextBlk = _
       igxShape.TextBlock.ChildTextBlocks.Item(iCount)
   With igxChildTextBlk.BlockFormat.LineFormat
       .Color = vbRed
       .Style = ixLineDashed
       .Width = 1
   End With
   ' Set the child text block margin
```

```
    igxChildTextBlk.TextMargin = 144
    ' Add text to the child text block
    igxChildTextBlk.Text = "Child Text Block " & iCount
    ' Give the child text block a yellow fill
    igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbYellow
Next iCount
MsgBox "View the diagram"
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 3
igxShape.DiagramObject.Width = 1440 * 1.5
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
```

{button ChildTextBlock object,JI(`igrafxrf.HLP',`ChildTextBlock_Object')}

TextRange Property

Syntax *ChildTextBlock*.**TextRange**([*First* As Long = 1], [*Last* As Long])

Data Type TextRange object (read-only, See Object Properties)

Description The TextRange property returns a TextRange object for the specified ChildTextBlock object. The purpose of this property is to provide control over a range of text within a ChildTextBlock.

The TextRange object lets you work with a range of text. The *First* and *Last* arguments specify the start and end positions of the text range. For example, specifying Paragraph1.TextRange(1,5) returns a TextRange that contains the first five characters of the paragraph. Specifying the property without providing the *First* and *Last* arguments returns a TextRange with all the characters in the paragraph. The *First* argument defaults to a value of 1, so to select from the first character of the paragraph only requires specifying the last character.

In addition, each Paragraph object contained within a ChildTextBlock has its own TextRange object that can be used to select either all or part of the paragraph.

Example The following example creates a shape in the active diagram, and sets up the shape's text block and two child text blocks. The TextRange property is then used to select the word 'child', and change its font to bold and red. The TextRange.Text property is printed to an Output window.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim igxTextRange As TextRange
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Set the shape's text block margins
With igxShape.TextBlock
    .BottomMargin = 0.1
    .LeftMargin = 0.1
    .RightMargin = 0.1
    .TopMargin = 0.1
End With
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineNormal
    .Width = 1
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "Main Text Block"
MsgBox "View the diagram"
' Add a ChildTextBlock at the left side of the main text block
' The size is in fixed units of twips; set to 360, or 1/4 inch
Set igxChildTextBlk = _
   igxShape.TextBlock.ChildTextBlocks.AddFixed(ixTextLeft, 360)
Set igxChildTextBlk = _
   igxShape.TextBlock.ChildTextBlocks.AddFixed(ixTextBottom, 360)
' Set the child text block's line format
For iCount = 1 To igxShape.TextBlock.ChildTextBlocks.Count
   Set igxChildTextBlk = _
```

```
        igxShape.TextBlock.ChildTextBlocks.Item(iCount)
    With igxChildTextBlk.BlockFormat.LineFormat
        .Color = vbRed
        .Style = ixLineDashed
        .Width = 1
    End With
    ' Add text to the child text block
    igxChildTextBlk.Text = "Child Text Block " & iCount
    ' Give the child text block a yellow fill
    igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbYellow
Next iCount
MsgBox "View the diagram"
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 3
igxShape.DiagramObject.Width = 1440 * 1.5
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
For iCount = 1 To igxShape.TextBlock.ChildTextBlocks.Count
    Set igxChildTextBlk = igxShape.TextBlock _
        .ChildTextBlocks.Item(iCount)
    Set igxTextRange = igxChildTextBlk.TextRange(1, 5)
    igxTextRange.Font.Bold = True
    igxTextRange.Font.Color = vbRed
    Application.Output igxTextRange.Text
Next iCount
MsgBox "View the diagram"
```

See Also

TextRange object

iGrafx API Object Hierarchy

{button ChildTextBlock object,JI(`igrafxrf.HLP',`ChildTextBlock_Object')}

ChildTextBlocks Object

The ChildTextBlocks object is a collection of individual ChildTextBlock objects. A ChildTextBlocks collection is associated with the TextBlock object of a Shape or ShapeClass object. Its purpose is to store and provide access to the individual ChildTextBlock objects that have been created for a TextBlock object.

This object provides the following functionality for working with ChildTextBlock objects:

- The ability to access any ChildTextBlock objects that have been created for a particular shape.
- The ability to determine how many ChildTextBlock objects are currently in the collection.
- The ability to add a new child text block to a shape's main text block.
- The ability to delete an existing child text block from a shape's main text block.

Properties, Methods, and Events

All of the properties, methods, and events for the ChildTextBlocks object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	AddFixed	
Count	AddPercentage	
Parent	Item	
	Remove	

Related Topics

ChildTextBlock object

iGrafx API Object Hierarchy

AddFixed Method

Syntax *ChildTextBlocks*.**AddFixed**(*Position* As IxChildTextPosition, *Size* As Long) As ChildTextBlock

Description The AddFixed method adds a new ChildTextBlock object to the ChildTextBlock collection, creating a “child text area” inside the main text block of the specified Shape object. This method adds a “fixed” size child text block, using unit of twips. Such a text block stays the same size, in twips, regardless of any resizing performed on the shape.

The *Position* argument specifies the location (right, left, top, or bottom) of the child text block within the shape’s main text block. The argument value is then applied to the Position property of the ChildTextBlock object that is created. The IxChildTextPosition constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixTextLeft
1	ixTextTop
2	ixTextRight
3	ixTextBottom

The *Size* argument specifies how big to make the child text block area. The size is specified in fixed units of twips (1440 twips = 1 inch). The argument value is then applied to the Size property of the ChildTextBlock object that is created.

To set the values of other ChildTextBlock properties, you must access the specific ChildTextBlock object once the new child text block has been created with this method.

Example The following example adds a shape to the active diagram. It then sets up shape’s text block with margins that are 1/10th of the shape’s width and height, and a border of a thin, solid blue line. A fixed size ChildTextBlock is then added on the left of 360 twips (1/4 inch), given a red dashed border line, filled with solid yellow, and given some text. The size of the main text block is then changed by altering the left margin.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
   Application.ShapeLibraries.Item(1).Item(1))
' Set the shape's text block margins
With igxShape.TextBlock
   .BottomMargin = 0.1
   .LeftMargin = 0.1
   .RightMargin = 0.1
   .TopMargin = 0.1
End With
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
   .Color = vbBlue
   .Style = ixLineNormal
   .Width = 1
End With
' Add text to the Text Block
```

```
igxShape.TextBlock.Text = "Main Text Block"
MsgBox "View the diagram"
' Add a ChildTextBlock at the left side of the main text block
' The size is in fixed units of twips; set to 360, or 1/4 inch
Set igxChildTextBlk = _
   igxShape.TextBlock.ChildTextBlocks.AddFixed(ixTextLeft, 360)
' Set the child text block's line format
With igxChildTextBlk.BlockFormat.LineFormat
   .Color = vbRed
   .Style = ixLineDashed
   .Width = 1
End With
' Add text to the child text block
igxChildTextBlk.Text = "Child Text Block 1"
' Give the child text block a yellow fill
igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbYellow
MsgBox "View the diagram"
' Change the left margin of the main text block
igxShape.TextBlock.LeftMargin = 0.5
MsgBox "View the diagram"
```

See Also

AddPercentage method

Remove method

ChildTextBlock.Position property

ChildTextBlock.SizeFixed property

ChildTextBlock.SizePercentage property

{button ChildTextBlocks object,JI(`igrafxrf.HLP',`ChildTextBlocks_Object')}

AddPercentage Method

Syntax *ChildTextBlocks*.**AddPercentage**(*Position* As IxChildTextPosition, *Size* As Double) As ChildTextBlock

Description The AddFixed method adds a new ChildTextBlock object to the ChildTextBlock collection, creating a “child text area” inside the main text block of the specified Shape object. This method adds a “variable” size child text block, specified as a percentage of the shape’s height or width, depending on the position. Such a text block grows or shrinks when the shape is resized to maintain the same percentage of the shape’s size.

The *Position* argument specifies the location (right, left, top, or bottom) of the child text block within the shape’s main text block. The argument value is then applied to the Position property of the ChildTextBlock object that is created. The IxChildTextPosition constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixTextLeft
1	ixTextTop
2	ixTextRight
3	ixTextBottom

The *Size* argument specifies how big to make the child text block area. The size is specified as a percentage of the shape’s size. For example, to add a text block that is 25% of the Shape’s width, specify a position of Left or Right, and set the Size argument to 0.25 (the valid range of values is >0.0 to 100.0, or greater if you want to specify a size of more than 100% of the shape’s size). The argument value is then applied to the Size property of the ChildTextBlock object that is created.

To set the values of other ChildTextBlock properties, you must access the specific ChildTextBlock object once the new child text block has been created with this method.

Example The following example adds a shape to the active diagram. It then sets up shape’s text block with margins that are 1/10th of the shape’s width and height, and a border of a thin, solid blue line. A variable size ChildTextBlock is then added on the left, at 25% of the shape’s width, given a red dashed border line, filled with solid yellow, and given some text. The size of the main text block is then changed by altering the left margin.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
   Application.ShapeLibraries.Item(1).Item(1))
' Set the shape's text block margins
With igxShape.TextBlock
   .BottomMargin = 0.1
   .LeftMargin = 0.1
   .RightMargin = 0.1
   .TopMargin = 0.1
End With
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
   .Color = vbBlue
```

```
    .Style = ixLineNormal
    .Width = 1
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "Main Text Block"
MsgBox "View the diagram"
' Add a ChildTextBlock at the left side of the main text block
' The size is a percentage of the width, set to 25%
Set igxChildTextBlk = _
    igxShape.TextBlock.ChildTextBlocks.AddPercentage(ixTextLeft, 25.0)
' Set the child text block's line format
With igxChildTextBlk.BlockFormat.LineFormat
    .Color = vbRed
    .Style = ixLineDashed
    .Width = 1
End With
' Add text to the child text block
igxChildTextBlk.Text = "Child Text Block 1"
' Give the child text block a yellow fill
igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbYellow
MsgBox "View the diagram"
' Change the left margin of the main text block
igxShape.TextBlock.LeftMargin = 0.5
MsgBox "View the diagram"
```

See Also

AddFixed method

Remove method

ChildTextBlock.Position property

ChildTextBlock.SizeFixed property

ChildTextBlock.SizePercentage property

{button ChildTextBlocks object,JI(`igrafxrf.HLP',`ChildTextBlocks_Object')}

Item Method

Syntax *ChildTextBlocks*.**Item**(*Index* As Integer) As ChildTextBlock

Description The Item method returns the ChildTextBlock object specified by the *Index* argument. If the index number is invalid, an error is returned. The result of this method must be assigned to a variable of type ChildTextBlock.

Example The following example creates a shape in the active diagram, and sets up its main text block. It then adds two child text blocks, and uses the Item method to set each child text block's line formatting, fill color, and text, and print the text of each child text block in the shape to an Output window. Finally, the shape is resized to see each text block area.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Set the shape's text block margins
With igxShape.TextBlock
    .BottomMargin = 0.1
    .LeftMargin = 0.1
    .RightMargin = 0.1
    .TopMargin = 0.1
End With
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineNormal
    .Width = 1
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "Main Text Block"
MsgBox "View the diagram"
' Add a ChildTextBlock at the left side of the main text block
' The size is in fixed units of twips; set to 360, or 1/4 inch
Set igxChildTextBlk = igxShape.TextBlock _
    .ChildTextBlocks.AddFixed(ixTextLeft, 360)
Set igxChildTextBlk = igxShape.TextBlock _
    .ChildTextBlocks.AddFixed(ixTextBottom, 360)
' Set the child text block's line format
For iCount = 1 To igxShape.TextBlock.ChildTextBlocks.Count
    Set igxChildTextBlk = igxShape.TextBlock _
        .ChildTextBlocks.Item(iCount)
    With igxChildTextBlk.BlockFormat.LineFormat
        .Color = vbRed
        .Style = ixLineDashed
        .Width = 1
    End With
    ' Add text to the child text block
    igxChildTextBlk.Text = "Child Text Block " & iCount
    ' Give the child text block a yellow fill
```

```
    igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbYellow
    Output "Child Text Block " & iCount & "'s text is: " _
         & igxChildTextBlk.Text
Next iCount
MsgBox "View the diagram"
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 3
igxShape.DiagramObject.Width = 1440 * 1.5
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
```

{button ChildTextBlocks object,JI(`igrafxrf.HLP',`ChildTextBlocks_Object')}

Remove Method

Syntax *ChildTextBlocks*.**Remove**(*Index* As Integer)

Description The Remove method removes a ChildTextBlock object from a TextBlock object. The *Index* argument specifies the ChildTextBlock to remove. If the index value is invalid, an error is returned.

Example The following example creates a shape in the active diagram, and sets up its main text block. It then adds two child text blocks, and uses the Item method to set each child text block's line formatting, fill color, and text. The shape is resized so each text block area can be seen. Finally, the first child text block is removed from the shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxChildTextBlk As ChildTextBlock
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1))
' Set the shape's text block margins
With igxShape.TextBlock
    .BottomMargin = 0.1
    .LeftMargin = 0.1
    .RightMargin = 0.1
    .TopMargin = 0.1
End With
' Set the text block's line format
With igxShape.TextBlock.BlockFormat.LineFormat
    .Color = vbBlue
    .Style = ixLineNormal
    .Width = 1
End With
' Add text to the Text Block
igxShape.TextBlock.Text = "Main Text Block"
MsgBox "View the diagram"
' Add a ChildTextBlock at the left side of the main text block
' The size is in fixed units of twips; set to 360, or 1/4 inch
Set igxChildTextBlk = igxShape.TextBlock _
   .ChildTextBlocks.AddFixed(ixTextLeft, 360)
Set igxChildTextBlk = igxShape.TextBlock _
   .ChildTextBlocks.AddFixed(ixTextBottom, 360)
' Set the child text block's line format
For iCount = 1 To igxShape.TextBlock.ChildTextBlocks.Count
   Set igxChildTextBlk = igxShape.TextBlock _
       .ChildTextBlocks.Item(iCount)
   With igxChildTextBlk.BlockFormat.LineFormat
        .Color = vbRed
        .Style = ixLineDashed
        .Width = 1
   End With
   ' Add text to the child text block
   igxChildTextBlk.Text = "Child Text Block " & iCount
   ' Give the child text block a yellow fill
```

```
    igxChildTextBlk.BlockFormat.FillFormat.FillColor = vbYellow
Next iCount
MsgBox "View the diagram"
' Make the shape larger and reset the position
igxShape.DiagramObject.Height = 1440 * 3
igxShape.DiagramObject.Width = 1440 * 1.5
igxShape.DiagramObject.Top = 1440
igxShape.DiagramObject.Left = 1440
MsgBox "View the diagram"
' Remove the first child text block
If ChildBlocks.Count > 0 Then
    igxShape.TextBlock.ChildTextBlocks.Remove(1)
End If
MsgBox "View the diagram"
```

See Also

AddFixed method

AddPercentage method

{button ChildTextBlocks object,JI(`igrafxrf.HLP',`ChildTextBlocks_Object')}

HeaderFooter Object

The HeaderFooter object is used to set the area size and the contents of the header and footer sections of a diagram. The HeaderFooter object is only accessed from the PageLayout object, which divides the page header and footer into three sections each. The six properties of the PageLayout object that correspond to the regions of the header and footer areas are:

- LeftHeader
- CenterHeader
- RightHeader
- LeftFooter
- CenterFooter
- RightFooter

You use the HeaderFooter object to designate the contents and formatting of each of the header and footer regions.

Properties, Methods, and Events

All of the properties, methods, and events for the HeaderFooter object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
BlockFormat		
Paragraphs		
Parent		
Text		
TextLF		
TextRange		

Related Topics

PageLayout object

iGrafx API Object Hierarchy

BlockFormat Property

Syntax *HeaderFooter*.**BlockFormat**

Data Type BlockFormat object (read-only, See Object Properties)

Description The BlockFormat property returns the BlockFormat object associated with the specified HeaderFooter object. The BlockFormat object controls the formatting of the text associated with a HeaderFooter object. Each HeaderFooter object has its own distinct BlockFormat object for controlling text formatting.

Example The following example adds some text to the left header region in two paragraphs, and displays how many paragraphs exist. It then uses the BlockFormat object to set the tab width to 1/4 inch, align the text vertically to the bottom of the header region, and set the fill of the region to a solid yellow.

```
' Dimension the variables
Dim igxTextGraphicObj As TextGraphicObject
Dim igxParagraph As Paragraph
Dim igxHeaderLeft As HeaderFooter
' Create a TextGraphic in the active diagram
Set igxTextGraphicObj = ActiveDiagram.DiagramObjects. _
    AddTextObject(1440, 1440, , , "A string of text")
' Get the left region of the header from the PageLayout object
Set igxHeaderLeft = ActiveDiagram.PageLayout.LeftHeader
' Set the text and alignment for the left header region
igxHeaderLeft.TextLF = "My Company" & Chr(13) & "My Division"
igxHeaderLeft.BlockFormat.HorizontalAlignment = ixHorizontalAlignLeft
' Verify that there are two paragraphs
MsgBox " The left header has " & igxHeaderLeft.Paragraphs.Count _
    & " paragraphs."
' Indent the second paragraph
igxHeaderLeft.BlockFormat.TabWidth = 360
igxHeaderLeft.BlockFormat.VerticalAlignment = ixVerticalAlignBottom
igxHeaderLeft.BlockFormat.FillFormat.FillType = ixFillSolid
igxHeaderLeft.BlockFormat.FillFormat.FillColor = vbYellow
igxHeaderLeft.Paragraphs(2).Indent
MsgBox "TabWidth of left header set to " _
    & igxHeaderLeft.BlockFormat.TabWidth & " twips, and " _
    & Chr(13) & "the second paragraph was indented." & Chr(13) _
    & "Return to the UI and view the header."
```

See Also BlockFormat object

iGrafx API Object Hierarchy

{button HeaderFooter object,JI(`igrafxrf.HLP',`HeaderFooter_Object')}

Paragraphs Property

Syntax *HeaderFooter*.**Paragraphs**

Data Type Paragraphs collection object (read-only, See Object Properties)

Description The Paragraphs property returns the Paragraphs collection associated with the specified HeaderFooter object. The Paragraphs object, through the Item method, provides access to the individual Paragragh objects.

Example The following example adds some text to the left header region in two paragraphs. It displays how many paragraphs exist, and then indents the second paragraph 1/4 inch.

```
' Dimension the variables
Dim igxTextGraphicObj As TextGraphicObject
Dim igxParagraph As Paragraph
Dim igxHeaderLeft As HeaderFooter
' Create a TextGraphic in the active diagram
Set igxTextGraphicObj = ActiveDiagram.DiagramObjects. _
    AddTextObject(1440, 1440, , , "A string of text")
' Get the left region of the header from the PageLayout object
Set igxHeaderLeft = ActiveDiagram.PageLayout.LeftHeader
' Set the text and alignment for the left header region
igxHeaderLeft.TextLF = "My Company" & Chr(13) & "My Division"
igxHeaderLeft.BlockFormat.HorizontalAlignment = ixHorizontalAlignLeft
' Verify that there are two paragraphs
MsgBox " The left header has " & igxHeaderLeft.Paragraphs.Count _
    & " paragraphs."
' Indent the second paragraph
igxHeaderLeft.BlockFormat.TabWidth = 360
igxHeaderLeft.Paragraphs(2).Indent
MsgBox "TabWidth of left header set to " _
    & igxHeaderLeft.BlockFormat.TabWidth & " twips, and " _
    & Chr(13) & "the second paragraph was indented." & Chr(13) _
    & "Return to the UI and view the header."
```

See Also Paragraph object

Paragraphs object

iGrafx API Object Hierarchy

{button HeaderFooter object,JI(`igrafxrf.HLP',`HeaderFooter_Object')}

TextRange Property

Syntax *HeaderFooter*.**TextRange**(*First* As Long, *Last* As Long)

Data Type TextRange object (read-only, See Object Properties)

Description The TextRange property returns a TextRange object for the specified HeaderFooter object. The purpose of this property is to provide control over a range of text within a HeaderFooter.

The TextRange object lets you work with a range of text. The *First* and *Last* arguments specify the start and end positions of the text range. For example, specifying Paragraph1.TextRange(1,5) returns a TextRange that contains the first five characters of the paragraph. Specifying the property without providing the *First* and *Last* arguments returns a TextRange with all the characters in the paragraph. The *First* argument defaults to a value of 1, so to select from the first character of the paragraph only requires specifying the last character.

Example The following example gets the diagram's left header section from the Pagelayout object. It adds some text and sets the horizontal alignment to "left". It then uses the TextRange object to select a range of text and change the font characteristics.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPageLayout As PageLayout
Dim igxHeaderLeft As HeaderFooter
Dim igxTextRange As TextRange
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the diagram's PageLayout object
Set igxPageLayout = ActiveDiagram.PageLayout
' Get the HeaderFooter object for the left header section
Set igxHeaderLeft = igxPageLayout.LeftHeader
' Add text to the left header and align it
igxHeaderLeft.Text = "My Company -- My Division"
igxHeaderLeft.BlockFormat.HorizontalAlignment = ixHorizontalAlignLeft
' Get "My Company" as a text range
Set igxTextRange = igxHeaderLeft.TextRange(1, 10)
' Set font properties for the text range
igxTextRange.Font.Color = vbRed
igxTextRange.Font.Bold = 1
MsgBox "Go to the interface and select File—Print Preview " _
    & "to view the change to the header area of the diagram"
```

See Also TextRange object

iGrafx API Object Hierarchy

{button HeaderFooter object,JI(`igrafxrf.HLP',`HeaderFooter_Object')}

Page Object

The Page object controls the mapping of output on a printed page to the positions of shapes and connectors on a diagram. For example, you might use the Page object and Pages collection to write a program that lays out objects and arranges them so they don't get broken in half by a page break.

This object provides locations of page edges and an ObjectRange property that can be used to help with diagram layout for printing. You can compare shape (or other diagram objects) locations against page edge locations, and then adjust either shape locations or page sizing accordingly.

The Page object is related to the PageLayout object in that the settings of the PageLayout properties affect the position and size of pages.

Properties, Methods, and Events

All of the properties, methods, and events for the Page object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Bottom		
Height		
Left		
ObjectRange		
Parent		
Right		
Top		
Width		

Related Topics

Pages object

PageLayout object

iGrafx API Object Hierarchy

Bottom Property

Syntax *Page*.**Bottom**

Data Type Long (read-only)

Description The Bottom property returns the location of the bottom of the specified Page object. The units are in twips (1440 twips = 1 inch).

Example The following example shows the use the the Bottom, Left, Right, Top, Width, and Height properties of the Page object. It also illustrates the relationship of the Page object with the PageLayout object

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPage As Page
Dim iCount As Integer
' Set the diagram's Page Layout and view properties
ActiveDiagram.PageLayout.PageOrder = ixDownThenAcross
ActiveDiagram.PageLayout.PaperSize = ixPaperSize10x14
ActiveDiagram.PageLayout.ScalingMode = ixScalingModeZoom
ActiveDiagram.Views.Item(1).DiagramView.Width = 1440 * 20
ActiveDiagram.PageLayout.OverlapAmount = 360
MsgBox "Page size is " _
    & CSng(ActiveDiagram.Pages.Item(1).Width / 1440) _
    & " x " & CSng(ActiveDiagram.Pages.Item(1).Height / 1440) _
    & Chr$(13) & "Page overlap is " _
    & CSng(ActiveDiagram.PageLayout.OverlapAmount / 1440)
' Change the page height to 5 inches
ActiveDiagram.PageLayout.PageHeight = 1440 * 5
MsgBox "Page Height changed to 5 inches"
' Create seven shapes in the active diagram in a row
For iCount = 1 To 7
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * iCount, 1440, _
        Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
        ActiveDiagram.DiagramObjects(iCount).Left = _
            ActiveDiagram.DiagramObjects(iCount - 1).Right + 980
    End If
Next iCount
' Display how many pages are in the diagram
MsgBox "This diagram has " & ActiveDiagram.Pages.Count & " pages." _
    & Chr$(13) & "There are " & ActiveDiagram.Pages.PagesAcross _
    & " pages across, and " & ActiveDiagram.Pages.PagesDown _
    & " pages down."
' Display the edge locations of the two pages
For iCount = 1 To ActiveDiagram.Pages.Count
    MsgBox "The location of page " & iCount & "'s edges are:" _
        & Chr$(13) & "Top: " & CSng(ActiveDiagram.Pages.Item(iCount) _
        .Top / 1440) & Chr$(13) & "Left: " _
        & CSng(ActiveDiagram.Pages.Item(iCount).Left / 1440) _
        & Chr$(13) & "Right: " _
        & CSng(ActiveDiagram.Pages.Item(iCount).Right / 1440) _
        & Chr$(13) & "Bottom: " _
```

```
        & CSng(ActiveDiagram.Pages.Item(iCount).Bottom / 1440)
Next iCount
' Add seven more shapes in a column
igxCurrentDOCount = ActiveDiagram.DiagramObjects.Count
For iCount = 1 To 7
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
       (ActiveDiagram.DiagramObjects(2).CenterX, _
       1440 * (iCount + 1), _
       Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
       ActiveDiagram.DiagramObjects(igxCurrentDOCount + iCount) _
           .Top = ActiveDiagram.DiagramObjects((igxCurrentDOCount _
           + iCount) - 1).Bottom + 980
    End If
Next iCount
' Display the page edge locations with Overlap at 0.25 inch
For iCount = 1 To ActiveDiagram.Pages.Count
   MsgBox "The location of page " & iCount & "'s edges are:" _
       & Chr$(13) & "Top: " & CSng(ActiveDiagram.Pages.Item(iCount) _
       .Top / 1440) & Chr$(13) & "Left: " _
       & CSng(ActiveDiagram.Pages.Item(iCount).Left / 1440) _
       & Chr$(13) & "Right: " _
       & CSng(ActiveDiagram.Pages.Item(iCount).Right / 1440) _
       & Chr$(13) & "Bottom: " _
       & CSng(ActiveDiagram.Pages.Item(iCount).Bottom / 1440)
Next iCount
ActiveDiagram.PageLayout.OverlapAmount = 0
MsgBox "Overlap amount set to: " _
   & CSng(ActiveDiagram.PageLayout.OverlapAmount / 1440)
' Display how many pages are in the diagram
MsgBox "This diagram has " & ActiveDiagram.Pages.Count & " pages." _
   & Chr$(13) & "There are " & ActiveDiagram.Pages.PagesAcross _
   & " pages across, and " & ActiveDiagram.Pages.PagesDown _
   & " pages down."
' Display the page edge locations with Overlap at 0
For iCount = 1 To ActiveDiagram.Pages.Count
   MsgBox "The location of page " & iCount & "'s edges are:" _
       & Chr$(13) & "Top: " & CSng(ActiveDiagram.Pages.Item(iCount) _
       .Top / 1440) & Chr$(13) & "Left: " _
       & CSng(ActiveDiagram.Pages.Item(iCount).Left / 1440) _
       & Chr$(13) & "Right: " _
       & CSng(ActiveDiagram.Pages.Item(iCount).Right / 1440) _
       & Chr$(13) & "Bottom: " _
       & CSng(ActiveDiagram.Pages.Item(iCount).Bottom / 1440)
Next iCount
```

See Also

Left property

Right property

Top property

{button Page object,JI(`igrafxrf.HLP',`Page_Object')}

Left Property

Syntax *Page*.**Left**

Data Type Long (read-only)

Description The Left property returns the location of the left edge of the specified Page object. The units are in twips (1440 twips = 1 inch).

Example Refer to the Example section of the Bottom property for a detailed code sample that shows the use of this property.

See Also Bottom property

Right property

Top property

{button Page object,JI(`igrafxrf.HLP',`Page_Object')}

ObjectRange Property

Syntax *Page*.**ObjectRange**

Data Type ObjectRange object (read-only, See Object Properties)

Description The ObjectRange property returns an ObjectRange object for the specified Page object. This property contains all of the DiagramObject objects that are located on the specified page.

Example The following example creates seven shapes in a horizontal row, and then displays the number of pages in the diagram. It then creates seven more shapes in a column, and again displays the number of pages. It then uses the Page object's ObjectRange property to display how many shapes are located on each page.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPage As Page
Dim igxCurrentDOCount As Integer
Dim igxPageObjRange As ObjectRange
' Set the diagram's page order to Down then Across
ActiveDiagram.PageLayout.PageOrder = ixDownThenAcross
' Create seven shapes in the active diagram in a row
For iCount = 1 To 7
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * iCount, 1440, _
        Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
        ActiveDiagram.DiagramObjects(iCount).Left = _
            ActiveDiagram.DiagramObjects(iCount - 1).Right + 980
    End If
Next iCount
' Display how many pages are in the diagram
MsgBox "This diagram has " & ActiveDiagram.Pages.Count & " pages." _
    & Chr$(13) & "There are " & ActiveDiagram.Pages.PagesAcross _
    & " pages across, and " & ActiveDiagram.Pages.PagesDown _
    & " pages down."
' Add seven more shapes in a column
igxCurrentDOCount = ActiveDiagram.DiagramObjects.Count
For iCount = 1 To 7
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (ActiveDiagram.DiagramObjects(2).CenterX, _
        1440 * (iCount + 1), _
        Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
        ActiveDiagram.DiagramObjects(igxCurrentDOCount + iCount) _
            .Top = ActiveDiagram.DiagramObjects((igxCurrentDOCount _
            + iCount) - 1).Bottom + 980
    End If
Next iCount
' Display how many objects are contained on each page using only
' the first argument of the Item method
For iCount = 1 To ActiveDiagram.Pages.Count
    Set igxPage = ActiveDiagram.Pages.Item(iCount)
    MsgBox "Page " & iCount & " contains " _
        & igxPage.ObjectRange.Count & " objects."
Next iCount
```

See Also

ObjectRange object

iGrafx API Object Hierarchy

{button Page object,JI(`igrafxrf.HLP',`Page_Object')}

Right Property

Syntax *Page*.**Right**

Data Type Long (read-only)

Description The Right property returns the location of the right edge of the specified Page object. The units are in twips (1440 twips = 1 inch).

Example Refer to the Example section of the Bottom property for a detailed code sample that shows the use of this property.

See Also Bottom property

Left property

Top property

{button Page object,JI(`igrafxrf.HLP',`Page_Object')}

Pages Object

The Pages object is a collection of individual Page objects. The Page object represents the image of a printed page of a diagram if it is printed using the current settings of the PageLayout object. A Pages collection is only associated with the Diagram object. Its purpose is to store and provide access to the individual Page objects. You can use the Pages collection and the Page object to map printed page output to positions of shapes and connectors on the diagram.

The Pages object provides the following functionality:

- The ability to access any Page objects in the collection.
- The ability to determine how many Page objects are currently in the collection.
- The ability to determine the number of pages, horizontally and vertically, that are specified for printing a diagram.

Properties, Methods, and Events

All of the properties, methods, and events for the Pages object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
PagesAcross		
PagesDown		
Parent		

Related Topics

Page object

PageLayout object

iGrafx API Object Hierarchy

Item Method

Syntax *Pages*.**Item**(*Index1* As Integer, [*Index2* As Integer = -1]) As Page

Description The Item method returns the Page object from the Pages collection identified by the specified index. The Item method can either take a page number (Index1 argument only), or an X and Y position where X is the horizontal position in the page matrix and Y is the vertical position in the page matrix.

The page number way of accessing the Pages collection changes depending on whether the pages are printed across then down or down then across (a setting which is set by the PageLayout object). For more information about pages, refer to the PageLayout object.

Example The following example shows several of the relationships between the Page and Pages objects and the PageLayout object. It shows the use of the Item method for specifying a page in both styles, and the use of the PagesAcross and PagesDown properties.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPage As Page
Dim igxCurrentDOCount As Integer
Dim igxPageObjRange As ObjectRange
' Set the diagram's page order to Down then Across
ActiveDiagram.PageLayout.PageOrder = ixDownThenAcross
' Create seven shapes in the active diagram in a row
For iCount = 1 To 7
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * iCount, 1440, _
        Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
        ActiveDiagram.DiagramObjects(iCount).Left = _
            ActiveDiagram.DiagramObjects(iCount - 1).Right + 980
    End If
Next iCount
' Display how many pages are in the diagram
MsgBox "This diagram has " & ActiveDiagram.Pages.Count & " pages." _
    & Chr$(13) & "There are " & ActiveDiagram.Pages.PagesAcross _
    & " pages across, and " & ActiveDiagram.Pages.PagesDown _
    & " pages down."
' Add seven more shapes in a column
igxCurrentDOCount = ActiveDiagram.DiagramObjects.Count
For iCount = 1 To 7
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (ActiveDiagram.DiagramObjects(2).CenterX, _
        1440 * (iCount + 1), _
        Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
        ActiveDiagram.DiagramObjects(igxCurrentDOCount + iCount) _
            .Top = ActiveDiagram.DiagramObjects((igxCurrentDOCount _
            + iCount) - 1).Bottom + 980
    End If
Next iCount
' Display how many pages are in the diagram
MsgBox "This diagram has " & ActiveDiagram.Pages.Count & " pages." _
    & Chr$(13) & "There are " & ActiveDiagram.Pages.PagesAcross _
    & " pages across, and " & ActiveDiagram.Pages.PagesDown _
```

```
    & " pages down."
' Display how many objects are contained on each page using only
' the first argument of the Item method
For iCount = 1 To ActiveDiagram.Pages.Count
    Set igxPage = ActiveDiagram.Pages.Item(iCount)
    MsgBox "Page " & iCount & " contains " _
        & igxPage.ObjectRange.Count & " objects."
Next iCount
' Display how many objects are contained on each page using both
' arguments of the Item method
For i = 1 To ActiveDiagram.Pages.PagesAcross
    For j = 1 To ActiveDiagram.Pages.PagesDown
        Set igxPage = ActiveDiagram.Pages.Item(i, j)
        MsgBox "Page " & i & ", " & j & " contains " _
            & igxPage.ObjectRange.Count & " objects."
    Next j
Next i
' Set the diagram's page order to Across then Down
ActiveDiagram.PageLayout.PageOrder = ixAcrossThenDown
' Display how many pages are in the diagram
MsgBox "This diagram has " & ActiveDiagram.Pages.Count & " pages." _
    & Chr$(13) & "There are " & ActiveDiagram.Pages.PagesAcross _
    & " pages across, and " & ActiveDiagram.Pages.PagesDown _
    & " pages down."
' Display how many objects are contained on each page using only
' the first argument of the Item method
For iCount = 1 To ActiveDiagram.Pages.Count
    Set igxPage = ActiveDiagram.Pages.Item(iCount)
    MsgBox "Page " & iCount & " contains " _
        & igxPage.ObjectRange.Count & " objects."
Next iCount
' Display how many objects are contained on each page using both
' arguments of the Item method
For i = 1 To ActiveDiagram.Pages.PagesAcross
    For j = 1 To ActiveDiagram.Pages.PagesDown
        Set igxPage = ActiveDiagram.Pages.Item(i, j)
        MsgBox "Page " & i & ", " & j & " contains " _
            & igxPage.ObjectRange.Count & " objects."
    Next j
Next i
' Change the Pages Across setting in the PageLayout object
ActiveDiagram.PageLayout.FitToPagesAcross = 3
' Display how many pages are in the diagram
MsgBox "This diagram has " & ActiveDiagram.Pages.Count & " pages." _
    & Chr$(13) & "There are " & ActiveDiagram.Pages.PagesAcross _
    & " pages across, and " & ActiveDiagram.Pages.PagesDown _
    & " pages down."
```

See Also PageLayout object

{button Pages object,JI(`igrafxrf.HLP',`Pages_Object')}

PagesAcross Property

Syntax *Pages*.**PagesAcross**

Data Type Integer (read-only)

Description The PagesAcross property returns the value of the PageLayout.FitToPagesAcross property, and indicates the number of pages that have been allocated horizontally for printing a diagram.

Example Refer to the Example section of the Item method for a code sample that uses the PagesAcross property.

See Also PagesDown property

PageLayout.FitToPagesAcross property

{button Pages object,JI(`igrafxrf.HLP',`Pages_Object')}

PagesDown Property

Syntax *Pages*.**PagesDown**

Data Type Integer (read-only)

Description The PagesDown property returns the value of the PageLayout.FitToPagesDown property, and indicates the number of pages that have been allocated vertically for printing a diagram.

Example Refer to the Example section of the Item method for a code sample that uses the PagesDown property.

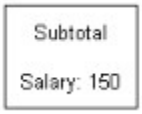
See Also PagesAcross property

PageLayout.FitToPagesDown property

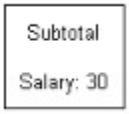
{button Pages object,JI(`igrafxrf.HLP',`Pages_Object')}

Legend Object

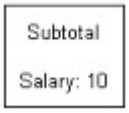
The Legend object displays subtotals or "accumulations" for custom data fields in objects in the diagram. For example, if you create a numeric custom data field called "Salary" in a diagram, you can specify when defining that data field the accumulation method. If you choose Sum for instance, the Legend object displays the sum of all the custom data values called "Salary". So if you have five shapes in the diagram with the custom data value "Salary" equal to 10, 20, 30, 40, and 50, the Legend looks like this:



If you change the accumulation method of the salary custom data field to "Mean", the Legend looks like this:



If you change the accumulation method to "Min", the Legend looks like this:



Properties, Methods, and Events

All of the properties, methods, and events for the Legend object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
DiagramObject		
FillFormat		
Font		
LineFormat		
Parent		

Related Topics

CustomDataDefinition object
CustomDataValue object
Field object
Fields object
iGrafx API Object Hierarchy

DiagramObject Property

Syntax *Legend*.**DiagramObject**

Data Type DiagramObject object (read-only, See Object Properties)

Description The DiagramObject property returns a DiagramObject object that is a Legend object (Type equals Legend). Some methods and properties of the DiagramObject object are not valid when the Type is Legend. Refer to the documentation of the DiagramObject object for more information.

Example The following example creates a legend in the active diagram. It then accesses the Legend object's extender, the DiagramObject, in order to set an object name. The Legend's object name is then displayed in a message box.

```
' Dimension the variables
Dim igxLegend As Legend
Dim igxDiagramObject As DiagramObject
' Create a shape on the active diagram
Set igxLegend = ActiveDiagram.DiagramObjects.AddLegend(1440, 1440)
' Get the DiagramObject object of the new shape
Set igxDiagramObject = igxLegend.DiagramObject
' Give the legend an object name
igxDiagramObject.ObjectName = "My Legend"
' Display the name of the legend
MsgBox "Legend.DiagramObject.ObjectName is " _
    & igxDiagramObject.ObjectName
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button Legend object,JI(`igrafxrf.HLP',`Legend_Object')}

FillFormat Property

Syntax *Legend*.**FillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The FillFormat property returns a FillFormat object for the specified Legend object. This object is used to set the fill formatting characteristics for a legend. The FillFormat object controls whether a fill is used, and if so, what type of fill (solid, pattern, or gradient), and the color or colors used.

There are numerous options for fill formats. The example below shows just one of many. Refer to the FillFormat object for more information.

Example The following example shows how the Legend object is used. Five shapes are created to represent employees. Two custom data definitions are added to the document, and values are filled in and displayed for each employee. Accumulation methods are set for each of the custom data definitions, and then a Legend is created that displays the custom data. This example shows how to set the legend's fill and line formatting, and its font.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField As Field
Dim igxCDataDefn As CustomDataDefinition
Dim igxLegend As Legend
' Create five shapes in the active diagram
For iCount = 1 To 5
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
         (1440, 1440 * iCount, _
         Application.ShapeLibraries.Item(1).Item(1))
    If (iCount > 1) Then
        ActiveDiagram.DiagramObjects(iCount).Top = _
            ActiveDiagram.DiagramObjects(iCount - 1).Bottom + 980
    End If
Next iCount
' Scale the view of the diagram
ActiveDocument.ActiveView.DiagramView.ZoomPercentage = 70
MsgBox "View the state of the diagram"
' Add text to each shape, indicating that the shapes
' represent employees
For iCount = 1 To 5
    ActiveDiagram.DiagramObjects.Item(iCount).Shape.Text _
        = "Employee " & Str(iCount)
Next iCount
MsgBox "View the state of the diagram"
' Add 2 custom data definitions: Salary and Age
Call ActiveDocument.CustomDataDefinitions.Add("Salary", _
    ixCustomDataFormatCurrencyBase)
Call ActiveDocument.CustomDataDefinitions.Add("Age", _
    ixCustomDataFormatGeneralBase)
' Create display fields for the 2 custom data definitions
' and display the description on the diagram
For iCount = 1 To 5
    Set igxField = ActiveDiagram.DiagramObjects(iCount) _
        .Fields.Add(ixFieldTextCustomData, "Salary", ixFieldAbove)
    ActiveDiagram.DiagramObjects(iCount).Fields.Item(1) _
        .ShowDescription = True
```

```
    Set igxField = ActiveDiagram.DiagramObjects(iCount) _
        .Fields.Add(ixFieldTextCustomData, "Age", ixFieldBelow)
    ActiveDiagram.DiagramObjects(iCount).Fields.Item(2) _
        .ShowDescription = True
Next iCount
MsgBox "View the state of the diagram"
' Set the accumulation method for each custom data definition
Set igxCDataDefn = ActiveDocument.CustomDataDefinitions.Item(1)
igxCDataDefn.AccumulationMethod = ixMean
Set igxCDataDefn = ActiveDocument.CustomDataDefinitions.Item(2)
igxCDataDefn.AccumulationMethod = ixRange
' Set values for each shape for the custom data
For iCount = 1 To 5
    Select Case iCount
        Case 1:
            ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(1).Value = 42500
            ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(2).Value = 33
        Case 2:
            ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(1).Value = 55750
            ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(2).Value = 41
        Case 3:
            ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(1).Value = 36000
            ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(2).Value = 29
        Case 4:
            ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(1).Value = 49400
             ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(2).Value = 45
        Case 5:
            ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(1).Value = 51800
             ActiveDiagram.DiagramObjects(iCount).CustomDataValues _
                .Item(2).Value = 38
    End Select
    MsgBox "View the state of the diagram"
Next iCount
' Create a Legend object and set its properties
Set igxLegend = ActiveDiagram.DiagramObjects.AddLegend(1440 * 5, 1440)
With igxLegend.FillFormat
    .FillType = ixFillSolid
    .FillColor = vbYellow
End With
With igxLegend.Font
    .Name = "Arial"
    .Bold = True
    .Color = vbBlack
End With
With igxLegend.LineFormat
    .Style = ixLineNormal
```

```
        .Color = vbGreen
        .Width = 2
    End With
    MsgBox "View the state of the diagram"
```

See Also

FillFormat object

iGrafx API Object Hierarchy

{button Legend object,JI(`igrafxrf.HLP',`Legend_Object')}

Font Property

Syntax *Legend*.**Font**

Data Type Font object (read-only, See Object Properties)

Description The Font property returns the Font object associated with the specified Legend object. You use this property to change the font, font style, font size, and font color of the text in a legend.

Example For an example that uses all the properties of the Legend object, refer to FillFormat property.

See Also Font object

iGrafx API Object Hierarchy

{button Legend object,JI(`igrafxrf.HLP',`Legend_Object')}

LineFormat Property

Syntax *Legend*.**LineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The LineFormat property returns the LineFormat object for the specified Legend object. This property allows you to change all of the line formatting attributes of the Legend object, such as color, style, and width.

Example For an example that uses all the properties of the Legend object, refer to FillFormat property.

See Also LineFormat object

iGrafx API Object Hierarchy

{button Legend object,JI(`igrafxrf.HLP',`Legend_Object')}

Callback Object

The Callback is an interface that you implement using, for example, a VBA class and the Implements keyword. It has one method, Execute, for which you must provide an implementation.

To implement the Callback interface, create a VBA class. Then, at the top of the class, type:

```
implements Callback
```

From the drop down menu above the code window, choose Callback. It has one method, Execute. In your implementation for the Execute method, write the code to be executed by your VBA class.

Your VBA class that implements Callback can be used by the following methods. All of these methods delay executing the code in your VBA class until some point in the future.

- Application.DoLater
- Application.RegisterTimer
- Document.DoAfterCurrentChangeBracket
- Document.DoAfterTopChangeBracket

Properties, Methods, and Events

All of the properties, methods, and events for the Callback object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
	Execute	

Execute Method

See the Callback Interface object documentation.

{button Callback object,JI(`igrafxrf.HLP',`Callback_Object')}

Diagram Object

The Diagram object is the API encapsulation of an iGrafx Professional diagram. A Diagram is the container for shapes, text, and all other objects that comprise the diagrams you develop with iGrafx Professional or iGrafx Process. The Diagram object is a property of the Document object. A document can contain multiple diagrams.

Many tasks and activities can be performed at the Diagram level (see the next section). The Diagram level also provides access to a number of other levels of the object hierarchy through its object properties. Some of the most important are:

- Any VB control object, through the AnyControls object
- Any DiagramObject
- Any CommandBar object
- All DataFieldTemplates and CustomDataValues objects
- The Document object that contains the diagram
- Any Department object
- All Entity objects
- All Layer objects

Properties, Methods, and Events

All of the properties, methods, and events for the Diagram object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
ActiveLayer	ActivateDiagram	Activate
AnyControls	ChangeDiagramType	AfterPrint
Application	CheckSpelling	AfterSaveAsWebPage
AsType	Copy	BeforeClick
CommandBars	CopyDiagram	BeforeClose
Departments	Cut	BeforeDoubleClick
DiagramObjects	DeleteDiagram	BeforeKeyDown
DiagramProtection	Export	BeforePrint
DiagramType	Find	BeforeRightClick
Document	FireUserEvent	Close
FullName	MakeObjectRange	ContextMenu
Guidelines	Paste	Deactivate
IndicatorFont	PasteDiagram	Delete
IntersectionColor	PasteLink	DiagramTypeChange
IntersectionStyle	PasteSpecial	GetInterface
Layers	PrintDiagram	LayerAdd
LinkIndicatorStyle	Refresh	LayerDelete
Name	ReplaceText	LayerRename
NextShapeNumber	UpdateFields	New
NoteIndicatorStyle		Open
OffPageConnectorFormat		PageLayoutChange
PageLayout		PropertyChange
Pages		Rename
Parent		Save
PermanentDiagram		SaveAsWebPage

PropertyLists
Selection
StartPointNames
VBAName
Views

SelectionChange
UserEvent

Activate Event

Syntax **Private Sub** *Diagram*_**Activate**()

Description The Activate event occurs when a Diagram is activated. You can use this event to perform particular actions when a Diagram is activated. “Activated” means that the window containing the Diagram has the focus, either because the user selected a different diagram, or because the Diagram.ActivateDiagram method was invoked in Visual Basic. The event is useful for such activities as:

- Modifying the interface when certain diagrams or Diagram types are activated.
- Initializing certain properties every time a particular diagarm or diagram type is activated.

Example The following example uses the AnyDiagram_Activate event to listen for diagram activations. When a diagram is activated, the event determines which diagram was activated using the ActiveDiagram object. It then changes the appearance of the toolbars based on which diagram was activated.

```
Public Sub Test()
    ' Dimension variables
    Dim igxDiagram1 As Diagram
    Dim igxDiagram2 As Diagram
    MsgBox "Click OK to create two new diagrams."
    ' If there's an error, skip that line
    On Error Resume Next
    ' Create two new diagrams
    Set igxDiagram1 = ActiveDocument.Diagrams.Add("Diagram A")
    Set igxDiagram2 = ActiveDocument.Diagrams.Add("Diagram B")
    MsgBox "Diagrams created.  Return to the diagram and try " _
   & Chr(13) & "selecting each of the diagrams." & _
   " Watch the appearance of " _
    & Chr(13) & "the toolbars as you select each diagram."
End Sub

' The Diagram_Activate event
Private Sub AnyDiagram_Activate()
    ' Check which diagram was activated
    Select Case ActiveDiagram.Name
        Case "Diagram A"
            ' For this diagram make the toolbar button large
            Application.CommandBars.LargeButtons = True
        Case "Diagram B"
            ' For this diagram don't make the toolbar buttons large
            Application.CommandBars.LargeButtons = False
    End Select
End Sub
```

See Also Deactivate event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

ActivateDiagram Method

Syntax *Diagram*.**ActivateDiagram**

Description The ActivateDiagram method makes the specified diagram the active diagram (gives that diagram the focus). Once this method is used to activate a particular diagram, you can access the diagram by using the Application.ActiveDiagram property.

Example The following example creates two new diagrams. It then activates the first diagram in the Diagrams collection, and displays its name in a message box.

```
' Dimension the variables
Dim igxDiagram As Diagram
' Create two new diagrams
Application.ActiveDocument.Diagrams.Add ("Alpha")
Application.ActiveDocument.Diagrams.Add ("Beta")
' Activate the first diagram in the Diagrams collection
Application.ActiveDocument.Diagrams.Item(1).ActivateDiagram
' Retrieve the current active diagram into the igxDiagram object
Set igxDiagram = Application.ActiveDiagram
' Display the name of the current active diagram
MsgBox "The name of the active diagram is " & igxDiagram.Name
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

ActiveLayer Property

Syntax *Diagram*.**ActiveLayer**

Data Type Layer object (read-only, See Object Properties)

Description The Layer property returns the currently active Layer object for the specified Diagram object.

Example The following example retrieves the name of the active layer and displays this name in a message box.

```
' Dimension the variables
Dim igxApp As Application
Dim igxDiagram As Diagram
Dim igxLayer As Layer
' Set the ixappApp variable to the current Application object
Set igxApp = Application.Application
' Set the Diagram variables to the new Diagram objects
Set igxDiagram = igxApp.ActiveDiagram
' Set the igxLayer variable to the Layer object
Set igxLayer = igxDiagram.ActiveLayer
' Display the name of the current active layer
MsgBox "The name of the active layer is " & igxLayer.Name
```

See Also Layer object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

AfterPrint Event

Syntax **Private Sub** *Diagram*_**AfterPrint**()

Description The AfterPrint event occurs after the command for the print of the diagram has been issued. This event can be useful if you have performed some modification to the diagram in the BeforePrint event, such as formatting or addition of a watermark, and you need to undo the changes.

Example The following example puts two colored shapes on the diagram, and then prints the diagram. The BeforePrint event changes all the shapes to white before printing. The AfterPrint event changes them back to their original colors after printing.

```
' Dimension an array to store shape colors
Private igxColors(32) As Long

' The main program
Sub Test()
    ' Dimension the variables
   Dim igxApp As Application
   Dim igxDiagram As Diagram
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Set the ixappApp variable to the current Application object
   Set igxApp = Application.Application
   ' Set the Diagram variables to the new Diagram objects
   Set igxDiagram = igxApp.ActiveDiagram
   ' Add two shapes to the diagram
   Set igxShape1 = igxDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = igxDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
   ' Set colors for the shapes
   igxShape2.FillColor = vbYellow
   igxShape1.FillColor = vbGreen
   ' Print the diagram
   If MsgBox("Diagram and shapes created. Print the diagram?", _
   vbYesNo) = vbYes Then
       ActiveDiagram.PrintDiagram
   End If
End Sub

' BeforePrint event, set all the shape colors to white
Private Sub AnyDiagram_BeforePrint()
    Dim igxDiagramObject As DiagramObject
   MsgBox "The BeforePrint event will now change " _
       & "all the shapes to white."
   ' Iterate through all the diagram objects
   For Index = 1 To ActiveDiagram.DiagramObjects.Count
       ' Check if it's a shape
       If ActiveDiagram.DiagramObjects.Item(Index).Type _
       = ixObjectShape Then
           ' Store each shape's color in our array
           igxColors(Index) = ActiveDiagram.DiagramObjects _
               .Item(Index).Shape.FillColor
           ' Change the shape's color to white
           ActiveDiagram.DiagramObjects.Item(Index) _
```

```
                .Shape.FillColor = vbWhite
        End If
    Next Index
End Sub

' AfterPrint event, change the shapes back to their original colors
Private Sub AnyDiagram_AfterPrint()
    Dim igxDiagramObject As DiagramObject
    MsgBox "The AfterPrint event will change the shapes" & _
        " back to their original colors."
    ' Iterate through all the objects in the diagram
    For Index = 1 To ActiveDiagram.DiagramObjects.Count
        ' Check if it's a shape
        If ActiveDiagram.DiagramObjects.Item(Index).Type _
        = ixObjectShape Then
            ' Retreive the original color from our array
            ActiveDiagram.DiagramObjects.Item(Index) _
                .Shape.FillColor = igxColors(Index)
        End If
    Next Index
End Sub
```

See Also BeforePrint event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

AfterSaveAsWebPage Event

Syntax **Private Sub** *Diagram*_**AfterSaveAsWebPage**()

Description The AfterSaveAsWebPage event occurs after a diagram is saved to disk as a web page. Diagrams are saved a web pages when the SaveAsWebPage method is invoked in Visual Basic, or when the user goes to the iGrafx Professional File menu and chooses File->Save As Web Page.

Example The following example monitors the AfterSaveAsWebPage event. If the user saves a diagram as a web page, the event presents the user with the option to launch their web browser.

```
Private Sub AnyDiagram_AfterSaveAsWebPage()
    If MsgBox("Web page saved. View it in your web browser?", _
   vbYesNo) = vbYes Then
        ' Replace this path with the path to your web browser
        Shell "c:\Program Files\Internet Explorer\ie40.exe"
   End If
End Sub
```

See Also SaveAsWebPage event

Document.SaveAsWebPage method

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

AnyControls Property

Syntax *Diagram*.**AnyControls**

Data Type AnyControls object (read-only, See Object Properties)

Description The AnyControls property returns the AnyControls object, which is at the application level of the object hierarchy. The AnyControls object can be used to establish an event synchronization with the any of the AnyControls objects from any level of the object model hierarchy. With this event synchronization, you can monitor such “any” events as AnyConnector, AnyDiagram, AnyShape, or AnyObject from other objects such as ShapeClass, which does not have direct access to the AnyControls object.

Example The following example demonstrates how to establish an event synch with the AnyObjects events. Once the event synch is established, you can respond to any of the AnyObject events through the igxEventsSynch variable. This code could be put in the ShapeClass and the SetupSynch procedure could be put in the Initialize event of the ShapeClass. The CancelSynch procedure code can be put in the ShapeClass.Terminate event.

```
Public WithEvents igxEventsSynch As DiagramObject

Sub SetupSynch()
    ' Set up the event synch with the AnyObject
   Set igxEventsSynch = _
       Application.ActiveDiagram.AnyControls.AnyObject
End Sub

Sub CancelSynch()
     ' Cancel the event synch with AnyObject
     Set igxEventsSynch = Nothing
End Sub
```

The next example sets up an Event synch with the AnyDiagram object. The example implements the Activate event, which is triggered every time a different diagram is selected, (brought to the front/gains the focus), such as when the Diagram.Activate method is used, or when the user clicks on a different diagram with the mouse.

```
' Dimension a Diagram Object that hears events
' The "WithEvents" keyword switches on the event listening feature
' of objects. This declaration is at the module level
' (not inside a Sub)
Public WithEvents MyAnyDiagram As Diagram

' The main program -- Run this Sub to establish the event
Public Sub EventTest()
    ' Create the Object. Event monitoring was already enabled
   ' when the object was declared
   Set MyAnyDiagram = Application.ActiveDiagram.AnyControls.AnyDiagram
   ' Confirm the setup with a message
   MsgBox "The event is now active. Return to the" _
       & " diagram and try selecting either diagram."
   ' Add a diagram so we have at least two for triggering the event
   Application.ActiveDocument.Diagrams.Add ("New Diagram")
End Sub

Private Sub AnyDiagram_Activate()
```

```
    MsgBox "A new diagram has been activated."
End Sub
```

See Also AnyControls object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

AsType Property

Syntax *Diagram*.**AsType**(*TypeName* As String) As Object

Data Type An Object of the type identified by the *TypeName* argument (read/write)

Description The AsType property allows you to add your own properties and methods to a document object, extending the object model. The properties and methods can be organized into one or more document types, using unique type names.

The *TypeName* argument is a string that names the custom type. It can be any string you choose, but it must be unique within the environment. In an integrated environment, other programmers may be accessing the document, and using it's AsType property. To prevent conflicting type names, it is suggested that you use your company or department name, followed by a descriptive type name (for example, "MyCompanyFactory")

Use the following basic steps to implement a custom property or method for the Document object.

1. Use Document.AsType ("my type name").MyMethod in your code.
2. Create a new Class, and design properties and methods in the class.
3. Set up the GetInterface event to check the TypeName string passed to it. If it matches your type name, set the Interface parameter equal to your new class.

When you use Document.AsType(*TypeName*) in your code, you gain access to the properties and methods that you have defined in the new Class. The Document.AsType property automatically fires an event called GetInterface. The GetInterface event can have one or more AsType's defined, each one distinguished by a unique type name. Based on the type name, the GetInterface event redirects execution to your new Class by setting the Interface parameter. If the Interface parameter is set to your new Class, the Class properties and methods become exposed to the Document object.

Example Using the AsType property, the GetInterface event, and VBA's support for Classes, you can extend key iGrafx objects. The first step to doing this is creating a VBA class. The following example shows the implementation of a simple class which has two properties—MainCourse, and Desert.

Insert a new class under ExtensionProject called Class1, and copy this block of code into it.

```
' Class
Public Property Get MainCourse() As String
   MainCourse = "Meatloaf"
End Property

Public Property Get Desert() As String
   Desert = "Cake"
End Property
```

These two blocks of code go in the ExtensionProject "This Application" code window.

```
' Run this to test the event
Sub Main()
    Dim igxDiagram As Diagram
   Set igxDiagram = ActiveDiagram
   MsgBox "The main course is " _
        & igxDiagram.AsType("Dinner").MainCourse
End Sub
```

```
' The GetInterface event is fired whenever the AsType method is used
' Based on the TypeName, redirect the interface to your custom class
Private Sub AnyDiagram_GetInterface(ByVal TypeName As String, Interface As
Object)
    ' If the broadcast type name is "Dinner", then set the interface
   If TypeName = "Dinner" Then
        ' TypeName gets broadcast everywhere, so we need to check if
        ' something else grabbed and set the Interface first
        If Interface Is Nothing Then
            Set Interface = New Class1
        Else
            MsgBox "ERROR: Someone else is using MyType"
        End If
   End If
End Sub
```

See Also GetInterface event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

BeforeClick Event

Syntax **Private Sub** *Diagram*_**BeforeClick**(ByVal *X* As Double, ByVal *Y* As Double, *Cancel* As Boolean)

Description The BeforeClick event occurs before a single mouse click event within a Diagram is handled by iGrafx Professional. The event allows you to perform some set of actions before the click event is processed. You can cancel the click event by setting the *Cancel* parameter to True within the event subroutine.

Example The following example displays in the Output window the coordinates of the mouse every time the user single clicks within a diagram named Diagram1.

```
Private Sub Diagram1_BeforeClick(ByVal X As Double, _
ByVal Y As Double, Cancel As Boolean)
   Output "Clicked at X:" & X & "  Y:" & Y
End Sub
```

See Also BeforeDoubleClick event

BeforeRightClick event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

BeforeClose Event

Syntax **Private Sub** *Diagram*_**BeforeClose**(*Cancel* As Boolean)

Description The BeforeClose event occurs before the specified Diagram is closed by iGrafx Professional. The event allows the you to perform some set of actions before the Diagram is closed. You can cancel the Close event by setting the *Cancel* parameter to True within the event subroutine.

Example The following example sets up the BeforeClose event to confirm closing a diagram before it actually closes. If the user choose "Cancel" the diagram is not closed.

```
' Adds an extra meaasure of safty before closing a diagram
Private Sub Diagram_BeforeClose(Cancel As Boolean)
   If MsgBox("About to close the diagram. Click OK to" _
   & Chr(13) & " continue, or Cancel to stop closing.", _
   vbExclamation + vbOKCancel) = vbCancel Then
       Cancel = True
   End If
End Sub
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

BeforeDoubleClick Event

Syntax **Private Sub** *Diagram*_**BeforeDoubleClick**(ByVal *X* As Double, ByVal *Y* As Double, *Cancel* As Boolean)

Description The BeforeDoubleClick event occurs before a double click within the specified diagram is handled by iGrafx Professional. The event allows you to perform some set of actions before the double click event is processed. You can cancel the double click event by setting the *Cancel* parameter to True within the event subroutine.

Note that when a user performs a double click, both the “click” and “double click” events are fired. Click events are fired from the first click of the mouse, and double click events are fired from the second click of the mouse.

Example The following example displays in the Output window the coordinates of the mouse every time the user double clicks within a diagram named Diagram1.

```
Private Sub Diagram1_BeforeDoubleClick(ByVal X As Double, _
ByVal Y As Double, Cancel As Boolean)
   Output "Clicked at X:" & X & "  Y:" & Y
End Sub
```

See Also BeforeClick event

BeforeRightClick event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

BeforeKeyDown Event

Syntax **Private Sub** *Diagram*_**BeforeKeyDown**(ByVal *KeyCode* As Integer, ByVal *Flags* As Long, *Cancel* As Boolean)

Description The BeforeKeyDown event occurs when a key is pressed on the keyboard. The event allows you to perform some set of actions before the keyboard input is processed. Typically, you will write code in this event to “listen” for particular keyboard input. You can cancel the event at any time by setting the *Cancel* parameter to True within the event subroutine. The *Cancel* paramter typically is used to ignore the input of certain keystrokes.

The *KeyCode* parameter is an integer value that specifies the virtual-key code of the key being pressed. Refer to the tables of key codes provided in the Application.BeforeKeyDown event topic.

The *Flags* parameter specifies the repeat count, scan code, extended-key flag, context code, previous key-state flag, and transition-state flag. Refer to the table provided in the Application.BeforeKeyDown event topic for values.

Example The following example displays the keycode and the flags value of keys as they are pressed by the user.

```
Private Sub Diagram_BeforeKeyDown(ByVal KeyCode As Integer, _
ByVal Flags As Long, Cancel As Boolean)
    ' Display the keycode and the flags in the output window
   Output KeyCode & ", " & Flags
End Sub
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

BeforePrint Event

Syntax **Private Sub** *Diagram*_**BeforePrint**()

Description The BeforePrint event occurs before a print event is processed by iGrafx Professional. The event allows you to perform some set of actions before the print event is processed, such as formatting or the addition of a watermark to the diagram before printing.

Example The following example puts two colored shapes on the diagram, and then prints the diagram. The BeforePrint event changes all the shapes to white before printing. The AfterPrint event changes them back to their original colors after printing.

```
' Dimension an array to store shape colors
Private igxColors(32) As Long

' The main program
Sub Test()
    ' Dimension the variables
   Dim igxApp As Application
   Dim igxDiagram As Diagram
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Set the ixappApp variable to the current Application object
   Set igxApp = Application.Application
   ' Set the Diagram variables to the new Diagram objects
   Set igxDiagram = igxApp.ActiveDiagram
   ' Add two shapes to the diagram
   Set igxShape1 = igxDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = igxDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
   ' Set colors for the shapes
   igxShape2.FillColor = vbYellow
   igxShape1.FillColor = vbGreen
   ' Print the diagram
   If MsgBox("Diagram and shapes created. Print the diagram?", _
   vbYesNo) = vbYes Then
       ActiveDiagram.PrintDiagram
   End If
End Sub

' BeforePrint event, set all the shape colors to white
Private Sub AnyDiagram_BeforePrint()
    Dim igxDiagramObject As DiagramObject
   MsgBox "The BeforePrint event will now change " _
       & "all the shapes to white."
   ' Iterate through all the diagram objects
   For Index = 1 To ActiveDiagram.DiagramObjects.Count
       ' Check if it's a shape
       If ActiveDiagram.DiagramObjects.Item(Index).Type _
       = ixObjectShape Then
           ' Store each shape's color in our array
           igxColors(Index) = ActiveDiagram.DiagramObjects _
               .Item(Index).Shape.FillColor
           ' Change the shape's color to white
           ActiveDiagram.DiagramObjects.Item(Index) _
               .Shape.FillColor = vbWhite
```

```
        End If
    Next Index
End Sub

' AfterPrint event, change the shapes back to their original colors
Private Sub AnyDiagram_AfterPrint()
    Dim igxDiagramObject As DiagramObject
    MsgBox "The AfterPrint event will change the shapes" & _
        " back to their original colors."
    ' Iterate through all the objects in the diagram
    For Index = 1 To ActiveDiagram.DiagramObjects.Count
        ' Check if it's a shape
        If ActiveDiagram.DiagramObjects.Item(Index).Type _
        = ixObjectShape Then
            ' Retreive the original color from our array
            ActiveDiagram.DiagramObjects.Item(Index) _
                .Shape.FillColor = igxColors(Index)
        End If
    Next Index
End Sub
```

See Also AfterPrint event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

BeforeRightClick Event

Syntax **Private Sub** *Diagram*_**BeforeRightClick**(ByVal *X* As Double, ByVal *Y* As Double, *Cancel* As Boolean)

Description The BeforeClick event occurs before a right mouse click within a Diagram is handled by iGrafx Professional. The event allows you to perform some set of actions before the right click event is processed. You can cancel the right click event by setting the *Cancel* parameter to True within the event subroutine.

Example The following example displayes the mouse coordinates whenever the user right clicks in a diagram named Diagram1.

```
Private Sub Diagram1_BeforeRightClick(ByVal X As Double, ByVal Y As Double,
Cancel As Boolean)
    MsgBox "Mouse was Right Clicked at X:" & X & "  Y:" & Y
End Sub
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

ChangeDiagramType Method

Syntax *Diagram*.**ChangeDiagramType**

Description The ChangeDiagramType method changes the diagram to either a Basic Diagram or a Process diagram (which are the two diagram types provided with iGrafx Professional and/or Process). The DiagramType argument specifies a DiagramType object, which is created from one of the two items in the Application.DiagramTypes collection.

Example The following example sets up the DiagramChangeType event to display a message when it's triggered. The Main( ) subroutine sets the diagram to one type, and then the other, which fires the event.

```
Public Sub Main()
    ' Dimension variables
   Dim igxTypeProcess As DiagramType
   Dim igxTypeBasicDiagram As DiagramType
   Dim igxDiagram As Diagram
   ' Set diagram types from the two built into the application
   Set igxTypeProcess = Application.DiagramTypes.Item(1)
   Set igxTypeBasicDiagram = Application.DiagramTypes.Item(2)
   ' Get the diagram object
   Set igxDiagram = ActiveDiagram
   MsgBox "Click OK to change the diagram type."
   ' Change the diagram type
   igxDiagram.ChangeDiagramType igxTypeProcess
   igxDiagram.ChangeDiagramType igxTypeBasicDiagram
End Sub

Private Sub AnyDiagram_DiagramTypeChange()
    MsgBox "The diagram type has been changed."
End Sub
```

See Also AsType method

DiagramType object

DiagramTypes object

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

CheckSpelling Method

Syntax *Diagram*.**CheckSpelling**

Description The CheckSpelling method checks the spelling of all text within the specified Diagram object. All textual elements of the diagram are checked, such as labels, text blocks, and notes. The method provides the same functionality as selecting Spelling from the Tools menu.

If a spelling error is found, the standard Microsoft Office Spelling dialog box is displayed, allowing the user to ignore the error, make corrections, add to the dictionary, or change all occurances of the spelling error.

Example The following example checks the spelling of all text within the active diagram.

```
Dim igxShape As Shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.Text = "This shape has some misspellled text."
' Use the CheckSpelling method to spell check the active diagram
Application.ActiveDiagram.CheckSpelling
MsgBox "Click OK to continue."
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Close Event

Syntax **Private Sub** *Diagram*_**Close**()

Description The Close event occurs when the Diagram object is closed either by the user or programmatically. Users can close a diagram by clicking the cross "X" button on the top right corner of the diagram window. A diagram can be closed from Visual Basic using the Diagram.Close method.

Example The following example uses the Close event to show how long a diagram has been open. It creates a new diagram in the document, and stores the time the diagram was created. When the diagram is closed, it displays the number of seconds the diagram was open. To try this example put this code in the Extension Project module called "ThisApplication".

```
' Dimension module variables
Private StartTime As Double
' Dimension a diagram variable that listens to events
Private WithEvents igxDiagram As Diagram

' The main program
Public Sub Main()
    ' Add a new diagram to the document
   Set igxDiagram = ActiveDocument.Diagrams.Add("Diagram B")
   ' Store the time the diagram was created
   StartTime = Timer
   MsgBox "Diagram opened. Return to the Diagram and Close " _
        & "the diagram."
End Sub

' The Close event
Private Sub igxDiagram_Close()
    ' Display the number of seconds the diagram was open
   MsgBox "Diagram closed. It was open for " & _
        Int(Timer - StartTime) & " seconds."
End Sub
```

See Also Open event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

CommandBars Property

Syntax *Diagram*.**CommandBars**

Data Type CommandBars collection (read-only, See Object Properties)

Description The CommandBars property returns the CommandBars collection object for the specified Diagram object. Each diagram contained in a document has its own CommandBars collection object. This means that the developer can customize the set of toolboxes associated with any specific diagram.

Example The following example retrieves the number of the CommandBar objects and displays this number in a Message Box. It then turns off display of color buttons, and then iterates through the collection making each command bar not visible. Then it again iterates through the collection, making each command bar visible. Finally it resets the buttons to display color.

```
' Display the number of command bar objects in the collection
MsgBox "There are currently " & ActiveDiagram.CommandBars.Count _
    & " command bar objects in the collection."
' Set all command bar objects to B&W buttons
ActiveDiagram.CommandBars.ColorButtons = False
For iCount = 1 To ActiveDiagram.CommandBars.Count
    ActiveDiagram.CommandBars.Item(iCount).Visible = False
    MsgBox "Made command bar " & iCount & " not visible"
Next iCount
' Restore the toolbars
For iCount = 1 To ActiveDiagram.CommandBars.Count
    ActiveDiagram.CommandBars.Item(iCount).Visible = True
    MsgBox "Made command bar " & iCount & " visible"
Next iCount
' Reset the buttons to color
ActiveDiagram.CommandBars.ColorButtons = True
```

See Also CommandBar object

CommandBars object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

ContextMenu Event

Syntax **Private Sub** *Diagram*_**ContextMenu**(*CommandBar* As CommandBar)

Description The ContextMenu event occurs when the specified Diagram object is “right-clicked” with the mouse, which opens the context menu. The event allows you to perform some set of actions before the context menu is opened.

The *CommandBar* parameter returns the context menu object (a CommandBar object), which can be used to alter the appearance of the menu before it pops up in the interface.

Example The following example demonstrates the ContextMenu event. To try it, put this code in a diagram code window, then go to the diagram. Right-Click on the blank diagarm surface and observe the altered captions. This example event alters the caption on the first three items in context menus.

```
Private Sub Diagram_ContextMenu(ByVal CommandBar As IXCommandBar)
    ' Alter the first three items on the context menu
   CommandBar.CommandBarItems.Item(1).Caption = "CUT!"
   CommandBar.CommandBarItems.Item(2).Caption = "COPY!"
   CommandBar.CommandBarItems.Item(3).Caption = "PASTE!"
End Sub
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Copy Method

Syntax *Diagram*.**Copy**

Description The Copy method copies all 'selected' DiagramObject objects and ObjectRange objects to the clipboard. The method is equivalent to the Edit->Copy command in the Edit menu. Any selected object is copied.

Use the Diagram.Paste method to paste the objects back into a diagram. Use the DiagramObject.Selected property or the Diagram.Selection property to select objects before using the Copy method. The Diagram.Selection property is an ObjectRange object that contains all the currently selected objects in the diagram. You can also use its methods to select objects.

Example The following example creates two shapes in the active diagram. It then selects and copies the objects. Then it pastes the objects back into the diagram at a different location.

```
' Add two shapes to the diagram
ActiveDiagram.DiagramObjects.AddShape 1440, 1440
ActiveDiagram.DiagramObjects.AddShape 1440 * 3, 1440
' Select all the shapes in the diagram
ActiveDiagram.Selection.AddAll ixObjectShape
' Copy the shapes
MsgBox "Click OK to copy and paste the objects to a new location."
ActiveDiagram.Copy
' Paste the shapes
ActiveDiagram.Paste 1440, 1440 * 3
MsgBox "Click OK to continue."
```

See Also

CopyDiagram method

Cut method

Paste method

Selection property

DiagramObject.Selected property

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

CopyDiagram Method

Syntax *Diagram*.**CopyDiagram**

Description The CopyDiagram method copies the entire Diagram object to the clipboard for the purpose of pasting it as a new component. This method is equivalent to clicking the "Copy" button in the Components dialog box.

Note In Visual Basic diagrams are not considered components, even though they appear in the Components dialog box. Also, Diagram objects do not appear in the Components collection object.

Example The following example copies and pastes the active diagram in the Components dialog box.

```
' Use CopyDiagram on the active diagram
MsgBox "Click OK to CopyDiagram the diagram."
ActiveDiagram.CopyDiagram
CommandBars.FindBuiltIn(ixFileMenu).CommandBarItems.Item(11) _
    .Command.Execute
MsgBox "Click OK to Paste the diagram"
' Send key to execute the Paste command
SendKeys "%P", True
MsgBox "Click OK to continue."
```

See Also Copy method

Cut method

Paste method

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Cut Method

Syntax *Diagram*.**Cut**

Description The Cut method cuts (or removes) all 'selected' DiagramObject objects and ObjectRange objects to the clipboard. The method is equivalent to the Edit->Cut command in the Edit menu. Any selected object is cut.

Use the Diagram.Paste method to paste the objects back into a diagram. Use the DiagramObject.Selected property or the Diagram.Selection property to select objects before using the Copy method. The Diagram.Selection property is an ObjectRange object that contains all the currently selected objects in the diagram. You can also use its methods to select objects.

Example The following example creates two shapes in the active diagram. It then selects the shapes and cuts them to the clipboard. Then it pastes them back into the diagram at a different location.

```
' Add two shapes to the diagram
ActiveDiagram.DiagramObjects.AddShape 1440, 1440
ActiveDiagram.DiagramObjects.AddShape 1440 * 3, 1440
' Select all the shapes in the diagram
ActiveDiagram.Selection.AddAll ixObjectShape
' Cut the shapes
MsgBox "Click OK to cut and paste the objects to a new location."
ActiveDiagram.Cut
' Paste the shapes at a new location
ActiveDiagram.Paste 1440, 1440 * 3
MsgBox "Click OK to continue."
```

See Also

Copy method

CopyDiagram method

Paste method

Selection property

DiagramObject.Selected property

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Deactivate Event

Syntax **Private Sub** *Diagram*_**Deactivate**()

Description The Deactivate event occurs when the Diagram object is deactivated. When another diagram is activated, the current diagram is deactivated, firing the event. Diagrams can be activated by the user, or from Visual Basic using the Activate method, causing the current diagram to be deactivated.

Example The following example gets the current diagram, and creates one more diagram. The Activate and Deactivate events are used to keep track of the time when any diagram is deactivated. When a diagram is activated, the time it was last deactivated is displayed.

```
' Dimension some module variables
Private WithEvents MyAnyDiagram As Diagram
Private Diagram1Time As Variant
Private Diagram2Time As Variant

Public Sub Main()
    ' Dimension the variables
   Dim igxDiagram1 As Diagram
   Dim igxDiagram2 As Diagram
   ' Get the current diagram
   Set igxDiagram1 = ActiveDiagram
   ' Add one new diagram
   Set igxDiagram2 = ActiveDocument.Diagrams.Add("Diagram E")
   ' Get the AnyDiagram object
   Set MyAnyDiagram = _
       Application.ActiveDocument.AnyControls.AnyDiagram
   ' Set the names of the diagrams for reference
   igxDiagram1.Name = "Diagram A"
   igxDiagram2.Name = "Diagram B"
   MsgBox "Diagram events ready.  Return to the diagram and" _
       & Chr(13) & " try clicking to activate each diagram."
End Sub

Private Sub MyAnyDiagram_Activate()
    ' When a diagram is activated, display the last time
   ' it was deactivated
   Select Case MyAnyDiagram.Name
       Case "Diagram A"
           MsgBox "Diagram A activated. It was deactivated " _
               & Diagram1Time
       Case "Diagram B"
           MsgBox "Diagram B activated. It was deactivated " _
               & Diagram2Time
   End Select
End Sub

Private Sub MyAnyDiagram_Deactivate()
    ' When a diagram is deactivated, store the time it happened
   Select Case MyAnyDiagram.Name
       Case "Diagram A"
           Diagram1Time = Now
       Case "Diagram B"
```

```
            Diagram2Time = Now
        End Select
    End Sub
```

See Also Activate event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Delete Event

Syntax **Private Sub** *Diagram*_**Delete**()

Description The Delete event occurs when a DiagramObject object is deleted from the specified diagram. A DiagramObject can be deleted by a user through the user interface, or programmatically, typically with the DiagramObjects.Item(Index).Delete method. You can delete a range of objects using the DiagramObjects.ObjectRange.Delete method.

Example The following example adds a shape to the active diagram. It then deletes the object from the diagram, which causes the Delete event to fire. The event displays a message.

```
' Dimension a diagram that listens to events
Private WithEvents igxDiagram As Diagram

' Main program
Public Sub Main()
    ' Dimension variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Set the diagram variable
   Set igxDiagram = ActiveDiagram
   ' Add two shapes to the diagram
   Set igxShape1 = igxDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Delete the shape
   MsgBox "Click OK to delete the object."
   igxDiagram.DiagramObjects.Item(1).DeleteDiagramObject
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagram_Delete()
    MsgBox "Shape deleted."
End Sub
```

See Also Copy method

Cut method

Paste method

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

DeleteDiagram Method

Topic Under Construction!!!

Syntax *Diagram*.**DeleteDiagram**

Description The DeleteDiagram method deletes the specified Diagram object from the Diagrams collection, and therefore, the current document.

Example The following example

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Departments Property

Syntax *Diagram*.**Departments**

Data Type Departments collection (read-only, See Object Properties)

Description The Departments property returns the Departments collection for the specified Diagram object. Each diagram contained in a document has its own Departments collection object, which stores the list of individual departments that have been created within the diagram.

Example The following example adds a department to the Diagram using the Departments object.

```
' Dimension the variables
Dim igxDepartments As Departments
' Set the igxDepartments variable to the Departments object
Set igxDepartments = ActiveDiagram.Departments
' Add a Department
MsgBox "Click OK to create a new department."
igxDepartments.AddDepartment "Test Department"
MsgBox "Click OK to continue."
```

See Also Department object

Departments object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

DiagramObjects Property

Syntax *Diagram*.**DiagramObjects**

Data Type DiagramObjects collection object (read-only, See Object Properties)

Description The DiagramObjects property returns the DiagramObjects collection for the specified Diagram object. This object contains all of the various DiagramObject objects that are currently contained in the diagram.

Example The following example adds some departments and shapes to the diagram. It then iterates through the DiagramObjects, and collects the names of the objects based on the object type, using the DiagramObjects property.

```
' Dimension the variables
Dim igxDiagramObjects As DiagramObjects
Dim sDepartments As String
Dim sShapes As String
' Set the igxDiagramObjects variable to the DiagramObjects object
Set igxDiagramObjects = ActiveDiagram.DiagramObjects
' Add some departments to the diagram
ActiveDiagram.Departments.AddDepartment ("Research")
ActiveDiagram.Departments.AddDepartment ("Production")
' Add some shapes to the diagram
Set igxShape1 = igxDiagramObjects.AddShape(1440, 1440)
Set igxShape2 = igxDiagramObjects.AddShape(1440, 1440 * 3)
igxShape1.Text = "Activity A"
igxShape2.Text = "Activity B"
' Iterate through all the diagram items
For Index = 1 To ActiveDiagram.DiagramObjects.Count
   ' Collect the object names based on object type
   Select Case igxDiagramObjects.Item(Index).Type
       Case ixObjectDepartment:
           sDepartments = sDepartments & _
               igxDiagramObjects.Item(Index).Department.Text & Chr(13)
       Case ixObjectShape:
           sShapes = sShapes & igxDiagramObjects.Item(Index) _
               .Shape.Text & Chr(13)
   End Select
Next Index
' Display the result
DS = Chr(13) & Chr(13) 'double space
MsgBox "The diagram contains these departments:" & DS & _
    sDepartments & DS & "And these shapes:" & DS & sShapes
```

See Also DiagramObject object

DiagramObjects object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

DiagramProtection Property

Syntax *Diagram*.**DiagramProtection**

Data Type IxDiagramProtection enumerated constant (read/write)

Description The DiagramProtection property locks a diagram so that no change can be made to it. Protected diagrams cannot be altered, either by Visual Basic or from the user interface. Objects on protected diagrams can still be copied, and their attributes read.

The IxDiagramProtection constant defines the valid values for this property, which are listed in the following table. The values True (Read-Only) and False (None) also can be used.

Value	Name of Constant
0	ixDiagramProtectNone
1	ixDiagramProtectReadOnly

Example The following example sets the active diagram as Read Only.

```
' Make the active diagram Read Only.
If MsgBox("Protect the diagram?", vbYesNo) = vbYes Then
    ActiveDiagram.DiagramProtection = ixDiagramProtectReadOnly
End If
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

DiagramType Property

Syntax *Diagram*.**DiagramType**

Data Type DiagramType object (read-only, See Object Properties)

Description The DiagramType property returns the DiagramType object for the specified Diagram object. The DiagramType object defines the type of diagram. IGrafx Professional and Process have two built-in diagram types: Basic diagram and Process diagram.

Example The following example retrieves the ShapeLibrary object from the DiagramType object of the active diagram.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxShapeLibrary As ShapeLibrary
' Set the igxDiagramType variable to the DiagramType object
Set igxDiagramType = ActiveDiagram.DiagramType
' Set the igxShapeLibrary variable to the ShapeLibrary object
' for this DiagramType object
Set igxShapeLibrary = igxDiagramType.ShapeLibrary
MsgBox "The Shape Library is called: " & _
    igxShapeLibrary.CollectionName
```

See Also DiagramType object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

DiagramTypeChange Event

Syntax **Private Sub** *Diagram*_**DiagramTypeChange**()

Description The DiagramTypeChange event occurs when the ‘type’ of the diagram is changed. The diagram type can be changed with the ChangeDiagramType method.

Example The following example sets up the DiagramTypeChange event to display a message when it's triggered. The Main( ) subroutine sets the diagram to one type, and then the other, which fires the event.

```
Public Sub Main()
    ' Dimension the variables
   Dim igxTypeProcess As DiagramType
   Dim igxTypeBasicDiagram As DiagramType
   ' Set diagram types from the two built into the application
   Set igxTypeProcess = Application.DiagramTypes.Item(1)
   Set igxTypeBasicDiagram = Application.DiagramTypes.Item(2)
   MsgBox "Click OK to change the diagram type."
   ' Change the diagram type
   ActiveDiagram.ChangeDiagramType igxTypeProcess
   ActiveDiagram.ChangeDiagramType igxTypeBasicDiagram
End Sub

Private Sub AnyDiagram_DiagramTypeChange()
   MsgBox "The diagram type has been changed."
End Sub
```

See Also ChangeDiagramType method

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Document Property

Syntax *Diagram*.**Document**

Data Type Document object (read-only, See Object Properties)

Description The Document property returns the Document object for the specified Diagram object. The property provides a way to traverse back up the object hierarchy to the Document level.

Example The following example retrieves the name of the Document and displays this name in a message box.

```
' Dimension the variables
Dim igxDocument As Document
' Get the active diagram's document object
Set igxDocument = ActiveDiagram.Document
' Display the name of the Document that contains the active diagram
MsgBox "The name of the Document is " & igxDocument.Name
```

See Also Document object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Export Method

Syntax Diagram.**Export**(*FileName* As String, [*SelectedOnly* As Boolean = False], [*ShowDialog* As Boolean = False]) As Boolean

Description The Export method exports a diagram to a new file format, with the option of saving only selected diagram objects. This method is equivalent to the Tools->Export Diagram… option in the Tools menu.

The *FileName* argument is a string specifying the path and file name of the exported diagram (for example, "C:\MyExportedDiagram.jpg"). The file extension (three-letter ending on the file name) is meaningful, especially if the *ShowDialog* argument is set to False (see below). The file extension determines the file type that is exported.

The *SelectedOnly* argument specifies whether only selected objects are included in the exported diagram. If the *SelectedOnly* argument is set to *True*, the diagram is saved containing only the objects which were selected before the export. The exported diagram has its margins moved in to surround the selected objects, resulting in a smaller diagram, just large enough to contain the selected objects. To select objects in the diagram, use the Selection property or the DiagramObject.selected property. If the *SelectedOnly* argument is set to *False*, all objects in the diagram are included in the exported file.

The *ShowDialog* argument specifies whether the Export dialog box is shown before exporting. If set to *False*, the file is exported without further input from the user, and the path and the file format is determined by the path and file extension used in the *FileName* argument. If set to *True*, the Export dialog box appears before the export is performed, allowing the user to change the path, file name, and export file type, if desired.

iGrafx Professional can export diagrams to a wide range of illustration, design and image file formats. For a complete list of file formats, go to the Tools menu, and select Tools->Export Diagram…->Save As Type (dropdown list).

Example The following example exports the selected objects in the diagram to a JPEG image file, and does not prompt the user for input before exporting. Be sure to change the path specified for the Export method to a location that is valid on your system.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
' Add three shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
' Select two of the shapes
ActiveDiagram.Selection.Add igxShape1.DiagramObject
ActiveDiagram.Selection.Add igxShape2.DiagramObject
' Export the selected shapes to a JPEG file
MsgBox "Click OK to save the selected objects as a JPEG image file."
ActiveDiagram.Export "E:\My Documents\exportedDiagram.jpg", True, False
MsgBox "JPEG file saved."
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Find Method

Syntax *Diagram*.**Find**(*ID* As Long) As DiagramObject

Description The Find method searches the diagram and returns a DiagramObject object. The search is based on the DiagramObject's ID number (the DiagramObject.ID property). The *ID* argument is the ID number to search for.

Error If the Find method fails to find the specified ID number, it produces a run-time error. Use error trapping if your code could potentially supply the *ID* argument with an ID number that does not exist in the diagram.

Example The following example creates two shapes and a connector line. It then stores the ID number of Shape 2, and later finds it using the Find method.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxConnector As ConnectorLine
   Dim igxObject As DiagramObject
   Dim MyID As Long
   ' Add two shapes and a connector line
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 4, 1440)
   Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, _
       ixDirWest)
   ' Store the ID number for Shape2
   MyID = igxShape2.DiagramObject.ID
   ' Trap the error if the item isn't found
   On Error GoTo NotFound
   MsgBox "Click OK to search for Shape 2"
   ' Find Shape2 using the ID we stored earlier
   Set igxObject = ActiveDiagram.Find(MyID)
   ' Change the text
   igxObject.Shape.Text = "This shape was found"
   MsgBox "Click OK to continue."
Exit Sub
' Display a message if the item wasn't found
NotFound:
   MsgBox "The specified ID was not found."
End Sub
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

FireUserEvent Method

Syntax *Diagram*.**FireUserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The FireUserEvent method fires the "UserEvent" for the specified diagram. You can use this functionality to send messages to any diagram that is listening to events.

You must specify an *EventIdentifier* argument (a string) to use for your event. You might choose to use something like your company name followed by the event name. You should choose a name that won't conflict with names picked by other developers.

You can pass one parameter to the event (the *Parameter* argument). This parameter is a Variant, so one logical choice is to pass a Class.

Then, you can write code in a UserEvent handler to perform some actions when your event fires. This code should be of the form:

```
If EventIdentifier = "<<Your identifier string>>" Then
   << Write your code here >>
End If
```

Example The following example defines a new user event called "ShowUsers". The *parameter* that gets passed is a class, which has one property called Count. The event handler displays the passed parameter's Count property.

The following code defines a simple class with one property. Create a new class below a diagram project called Class1 and copy this code into it.

```
' Class1
' It contains one property, read only
Public Property Get Count() As Long
   Count = 25
End Property
```

The following is the main program. Copy this, and the UserEvent subroutine, into the diagram project code window

```
' Run this subroutine to test the event
Public Sub Main()
    ' Create a new Class1 object
   Dim MyClass1 As New Class1
   ' Fire the UserEvent
   ActiveDiagram.FireUserEvent "ShowUsers", MyClass1
End Sub

' This event handler runs every time any FireUserEvent method
' is used in the system
Private Sub Diagram_UserEvent(ByVal EventIdentifier As String, ByVal Parameter
As Variant)
    ' Check if the Identifier string is the one we want
   If EventIdentifier = "ShowUsers" Then
        ' Redirect to Class1
       MsgBox "The number of users is " & Parameter.Count
   End If
End Sub
```

See Also UserEvent event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

GetInterface Event

Syntax **Private Sub** *Diagram*_**GetInterface**(ByVal *TypeName* As String, *Interface* As Object)

Description The GetInterface event occurs when the Diagram.AsType property is used. The AsType property allows you to add your own properties and methods to a Diagram object, extending the object model. The properties and methods can be organized using unique type names.

The *TypeName* argument is a string that distinguishes the custom type. It can be any string the programmer chooses, but it must be unique within the environment. In an integrated environment, other programmers may be accessing the diagram, and using it's AsType property. To prevent conflicting type names, it is suggested that you use your company or department name, followed by a descriptive type name (for example, "MyCompanyFactory").

Use the following basic steps to implement a custom property or method for a Diagram object.

1. Use Diagram.AsType ("my type name").MyMethod in your code.
2. Create a new Class, and design properties and methods in the class.
3. Set up the GetInterface event to check the TypeName string passed to it. If it matches your type name, set the Interface parameter equal to your new class.

When you use Diagram.AsType(*TypeName*) in your code, you gain access to the properties and methods that you have defined in the new Class. The Diagram.AsType property automatically fires an event called GetInterface. The GetInterface event can have one or more AsType's defined, each one distinguished by a unique type name. Based on the type name, the GetInterface event redirects execution to your new Class by setting the Interface parameter. If the Interface parameter is set to your new Class, the Class properties and methods become exposed to the Diagram object.

Notes When you extend an iGrafx Professional object using the GetInterface event, you need to keep in mind that other developers may be using this event also. To be a good citizen, you should do the following:

- Be sure to pick a name that is likely to be unique for your AsType name. In the example above, "MyType" is too generic and it is possible that another developer could use the same name. Instead, follow the convention of using your name or your company name, a period, and a description of the type. For example, if you were writing a type that extended Application to add additional internet capabilities, and your company name was "Micrografx", you could name your AsType name "Micrografx.InternetExtension".
- When you write code in the GetInterface event, keep it simple. You should not do any time consuming operation in the GetInterface event such as querying a database or displaying a dialog box.
- When you write code in the GetInterface event, be aware of the current state of the Interface parameter. In the example above, this is illustrated by the code fragment "Interface Is Nothing". If this code fragment evaluates to true, then it is safe to Set the interface to your class. If this code fragment evaluates to false then someone else has already responded to the event and set the interface to their class. If this condition arises, you should try changing your AsType name.

Example Using the AsType property, the GetInterface event, and VBA's support for Classes, you can extend key iGrafx objects. The first step to doing this is creating a VBA class. The following example shows the implementation of a simple class which has two properties—MainCourse, and Desert.

Insert a new class under ExtensionProject called Class1, and copy this block of code into it.

```
' Class
Public Property Get MainCourse() As String
   MainCourse = "Meatloaf"
End Property
```

```
Public Property Get Desert() As String
    Desert = "Cake"
End Property
```

These two blocks of code go in the ExtensionProject "This Application" code window.

```
' Run this to test the event
Sub Main()
    Dim igxDiagram As Diagram
    Set igxDiagram = ActiveDiagram
    MsgBox "The main course is " _
        & igxDiagram.AsType("Dinner").MainCourse
End Sub

' The GetInterface event is fired whenever the AsType property is used
' Based on the TypeName, redirect the interface to your custom class
Private Sub AnyDiagram_GetInterface(ByVal TypeName As String, Interface As
Object)
    ' If the broadcast type name is "Dinner", then set the interface
    If TypeName = "Dinner" Then
        ' TypeName gets broadcast everywhere, so we need to check if
        ' something else grabbed and set the Interface first
        If Interface Is Nothing Then
            Set Interface = New Class1
        Else
            MsgBox "ERROR: Someone else is using MyType"
        End If
    End If
End Sub
```

See Also AsType property

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Guidelines Property

Syntax *Diagram*.**Guidelines**

Data Type Guidelines collection object (read-only, See Object Properties)

Description The Guidelines property returns the Guidelines collection for the specified Diagram object. This object contains all of the guidelines that have been created for a diagram.

Each diagram contained in a document has its own Guidelines collection object. This means that the developer can customize the set of guidelines associated with any specific diagram.

Example The following example creates a horizontal and a vertical guideline, one 2 inches down and one two inches from the left.

```
MsgBox "Click OK to create guidlines that cross at the 2 inch mark."
ActiveDiagram.Guidelines.Add 2880, ixGuidelineHorizontal
ActiveDiagram.Guidelines.Add 2880, ixGuidelineVertical
MsgBox "Click OK to continue."
```

See Also Guideline object

Guidelines object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

IndicatorFont Property

Syntax *Diagram*.**IndicatorFont**

Data Type Font object (read-only, See Object Properties)

Description The IndicatorFont property returns the Font object that controls Note and Link indicators within the specified Diagram object. The property provides control over the font characteristics of all Note indicators and Link indicators.

Example The following example adds a Note to a shape. It then uses the IndicatorFont property to set the font size of the Note indicator to 14 points (any Link indicators also would be affected).

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxFont As Font
' Add two shapes and a connector line
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add a Note to Shape 1
igxShape1.Note.Text = "This is a note."
ActiveDiagram.NoteIndicatorStyle.Text = "FYI"
MsgBox "Click OK to increase the indicator font size."
' Get the indicator font and increase the size
Set igxFont = ActiveDiagram.IndicatorFont
igxFont.Size = 14
MsgBox "Click OK to continue."
```

See Also Font object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

IntersectionColor Property

Syntax *Diagram*.**IntersectionColor**

Data Type Color (read/write)

Description The IntersectionColor property specifies the color of connector line intersections, when the intersection style is set to a value other than ixInterectionNone.

Example The following example creates three shapes and two connector lines. The connector lines are routed so they intersect. Then the intersection style and color are changed.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
' Add several objects to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 3)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
' Add intersecting connector lines
Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape2, _
    ixDirEast)
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape3, _
    ixDirWest)
' Change the intersection appearance
MsgBox "Click OK to indicate intersections with a red square."
ActiveDiagram.IntersectionStyle = ixIntersectionLargeSquare
ActiveDiagram.IntersectionColor = vbRed
MsgBox "Click OK to continue."
```

See Also IntersectionStyle property

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

IntersectionStyle Property

Syntax *Diagram*.**IntersectionStyle**

Data Type IxIntersectionStyle enumerated constant (read/write)

Description The IntersectionStyle property specifies the style for connector line intersections.

The IxIntersectionStyle constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixInterectionNone
1	ixInterectionCircle
2	ixInterectionSmallSquare
3	ixInterectionLargeSquare

Example The following example creates three shapes and two connector lines. The connector lines are routed so they intersect. Then the intersection style is set to each possible value, and the color is set to vbRed.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
' Add several objects to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 3)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
' Add intersecting connector lines
Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape2, _
    ixDirEast)
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape3, _
    ixDirWest)
' Cycle through all intersection styles and make the color vbRed
ActiveDiagram.IntersectionColor = vbRed
For iCount = 0 To 3
    ActiveDiagram.IntersectionStyle = iCount
    MsgBox "Click OK to continue."
Next iCount
```

See Also IntersectionColor property

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

LayerAdd Event

Syntax **Private Sub** *Diagram*_**LayerAdd**(*Layer* As Layer)

Description The LayerAdd event occurs when a layer is added to the Diagram object. The event is useful when you want to perform some action, such as updating a dialog, when a layer has been added to a diagram. The *Layer* parameter is the Layer object that was added to the diagram.

Example The following example adds one layer to the diagram. Adding the layer fires the LayerAdd event, which displays a message indicating the name of the layer that was added.

```
Public Sub Main()
   ActiveDiagram.Layers.Add "MyLayer"
   MsgBox "Click OK to continue."
End Sub

Private Sub Diagram_LayerAdd(ByVal Layer As Layer)
   MsgBox "A Layer called " & Layer.Name & " has been added."
End Sub
```

See Also LayerDelete event

LayerRename event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

LayerDelete Event

Syntax **Private Sub** *Diagram*_**LayerDelete**(ByVal *LayerIndex* As Integer)

Description The LayerDelete event occurs when a layer is deleted from a Diagram object. The event is useful when you want to perform some action, such as updating a dialog, when a layer has been deleted from a diagram. The *LayerIndex* parameter is an integer containing the index into the Layers collection pointing to the layer being deleted.

Example The following example adds a layer to the active diagram. It then deletes the layer, which fires the LayerDelete event.

```
Public Sub Main()
    ActiveDiagram.Layers.Add "Layer C"
   ActiveDiagram.Layers.Item(2).Delete
   MsgBox "Click OK to continue."
End Sub

Private Sub Diagram_LayerDelete(ByVal LayerIndex As Integer)
   MsgBox "Layer #" & LayerIndex & " was deleted."
End Sub
```

See Also LayerAdd event

LayerRename event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

LayerRename Event

Syntax **Private Sub** *Diagram*_**LayerRename**(*OldName* As String, *Layer* As Layer)

Description The LayerRename event occurs when a Layer object is renamed in the specified diagram. The event allows you to perform some set of actions when the layer rename event is handled by iGrafx Professional. The *OldName* parameter is the name of the layer before it was renamed. The *Layer* parameter is the Layer object that had just been renamed (its Name property contains the new name).

Example The following example renames a layer in the active diagram, which fires the LayerRename event. Using the arguments of the LayerRename event, a message is displayed that provides both the old name and new name of the layer.

```
Public Sub Main()
   MsgBox "Click OK to rename the first layer."
   ActiveDiagram.Layers.Item(1).Name = "Layer A"
   MsgBox "Try renaming layers, which fires the event."
End Sub

Private Sub Diagram_LayerRename(ByVal OldName As String, ByVal Layer As Layer)
   MsgBox "Layer " & OldName & " has been renamed to " & Layer.Name
End Sub
```

See Also LayerAdd event

LayerDelete event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Layers Property

Syntax *Diagram*.**Layers**

Data Type Layers collection (read-only, See Object Properties)

Description The Layers property returns the Layers collection for the specified Diagram object. Each diagram contained in a document has its own Layers collection object.

Example The following example adds one layer to the diagram, and then displays the names of all the layers on the diagram, using the Layers object.

```
' Dimension the variables
Dim sString As String
MsgBox "Click OK to add a layer to the diagram"
ActiveDiagram.Layers.Add "Layer B"
For Index = 1 To ActiveDiagram.Layers.Count
   sString = sString & ActiveDiagram.Layers.Item(Index).Name _
       & Chr(13)
Next Index
MsgBox "The diagram contains these layers: " & Chr(13) & Chr(13) _
   & sString
```

See Also Layer object

Layers object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

LinkIndicatorStyle Property

Syntax *Diagram*.**LinkIndicatorStyle**

Data Type LinkIndicatorStyle object (read-only, See Object Properties)

Description The LinkIndicatorStyle property returns the LinkIndicatorStyle object for the specified Diagram object.

Example The following example uses the LinkInicatorStyle.Style property to change the appearance of the link on the shape to be a link icon.

```
' Dimension the variables
Dim igxDiagramObjects As DiagramObjects
Dim igxShape As Shape
' Get the DiagramObjects object
Set igxDiagramObjects = ActiveDiagram.DiagramObjects
' Add a shape to the diagram
Set igxShape = igxDiagramObjects.AddShape(1440, 1440)
' Add a link to the shape
igxShape.Links.AddDiagramLink "Diagram B"
MsgBox "Click to change the diagram's LinkIndicator to a Link icon."
' Change the LinkIndicatorStyle to an icon
ActiveDiagram.LinkIndicatorStyle.Style = ixLinkIcon
MsgBox "Click OK to continue."
```

See Also LinkIndicatorStyle object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

MakeObjectRange Method

Syntax *Diagram*.**MakeObjectRange** As ObjectRange

Description The MakeObjectRange method creates a blank object range that can be used to create a custom object range.

Example The following example creates an ObjectRange object, and two shapes. It adds the shapes to the ObjectRange, and then changes the fill color of the ObjectRange.

```
' Dimension the variables
Dim igxObjectRange As ObjectRange
Dim igxShape1 As Shape
Dim igxShape2 As Shape
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add two shapes to the diagram
MsgBox "Click OK to add two shapes to the diagram."
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
' Add the shapes to the ObjectRange
MsgBox "Click OK to add the shapes to the ObjectRange."
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
' Change the fill color of the ObjectRange to blue
MsgBox "Now click OK to change the ObjectRange to blue."
igxObjectRange.FillFormat.FillColor = vbBlue
MsgBox "Click OK to continue."
```

See Also ObjectRange object

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

NextShapeNumber Property

Syntax *Diagram*.**NextShapeNumber**

Data Type Long (read-only)

Description The NextShapeNumber property returns the next shape number that will be used when a new shape is added to the specified diagram. The property returns a Long value. Every time a new shape is added to the diagram, this property is incremented by one. The value of this property is not decremented if shapes are deleted.

Example The following example uses the NextShapeNumber property to determine what number is currently the highest shape number.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
' Add several objects to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 3)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
' Add intersecting connector lines
Set igxConnector1 =ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape2, _
    ixDirEast)
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape3, _
    ixDirWest)
' Display the value of the highest shape number in the diagram
MsgBox "The highest shape number on the diagram is: " _
    & ActiveDiagram.NextShapeNumber - 1
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

New Event

Syntax **Private Sub** *Diagram*_**New**()

Description The New event occurs when a diagram is created. The event is useful if a diagram is created from a template that has code in the New event. Because the New event works at the diagram level, it only works if it is saved in a template, and a new diagram is created from the template.

Example The following example displays a message when a diagram is created using a template. To try this example, follow the five steps listed below:

1. Open a template (.igt) file by choosing **File->Template**…
2. Go to the Visual Basic editor by choosing **Tools->Visual Basic->Edit Code…**
3. Copy this code (or your own New event code) into the code window

```
Private Sub Diagram_New()
    MsgBox "A diagram was created using this template."
End Sub
```

4. Save the template by choosing **File->Template**…
5. Create a new diagram from the template you just modified by choosing **File->New->From Template…** This fires the New event.

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

NoteIndicatorStyle Property

Syntax *Diagram*.**NoteIndicatorStyle**

Data Type NoteIndicatorStyle object (read-only, See Object Properties)

Description The NoteIndicatorStyle property returns the NoteIndicatorStyle object for the specified Diagram object.

Example The following example creates a shape with a note, and then sets the note object's shadow attribute.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxNoteIndicatorStyle As NoteIndicatorStyle
' Add a shape to the diagarm
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a note to the shape
igxShape.Note.Text = "This is a note."
' Set the igxNoteIndicatorStyle variable
Set igxNoteIndicatorStyle = ActiveDiagram.NoteIndicatorStyle
MsgBox "Set the shadow attribute of the diagram's note object."
' Set the shadow attribute of the note object
igxNoteIndicatorStyle.Shadow = True
MsgBox "Click OK to continue"
```

See Also NoteIndicatorStyle object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

OffPageConnectorFormat Property

Syntax *Diagram*.**OffPageConnectorFormat**

Data Type OffPageConnectorFormat object (read-only, See Object Properties)

Description The OffPageConnectorFormat property returns the OffPageConnectorFormat object for the specified Diagram object.

Example The following example creates two shapes, each one on a separate page. It then sets the OffPageConnectorFormat.AutomaticConnectors property to True, which makes the connector line display as an off page connector.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxOffPageConnectorFormat As OffPageConnectorFormat
' Add a shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add another shape, but on the next page to the right
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 9, 1440)
' Connect the shapes with a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Set the igxOffPageConnectorFormat variable
Set igxOffPageConnectorFormat = ActiveDiagram.OffPageConnectorFormat
igxOffPageConnectorFormat.AutomaticConnectors = False
MsgBox "Click OK to format the line as an off page connector."
' Set automatic off page connectors True
igxOffPageConnectorFormat.AutomaticConnectors = True
MsgBox "Click OK to continue."
```

See Also OffPageConnectorFormat object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Open Event

Syntax **Private Sub** *Diagram*_**Open**()

Description The Open event occures when a diagram is opened. Opening a diagram is different from activating a diagram, which occurs when a loaded diagram gains the focus. The Open event only fires if the event code is saved to disk with a diagram. Upon loading a document from disk that contains the diagram, the diagram is then opened, which fires the event.

Example The following example displays a custom copyright message when a diagram is opened. To try this example, use the following steps.

1. Create a new diagram by choosing **File->New**
2. Open the Visual Basic code window for the diagram and copy this code into it

```
Private Sub Diagram_Open()
    MsgBox "My Diagram, Copyright (c) 1999, My Company, Inc." _
        , , "Copyright Notice"
End Sub
```

3. Save the diagram (the document) by choosing **File->Save**
4. Close the document by choosing **File->Close**
5. Reload the document by choosing **File->Open**. When the document loads, the Open event for the diagram will fire.

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

PageLayout Property

Syntax *Diagram*.**PageLayout**

Data Type PageLayout object (read-only, See Object Properties)

Description The PageLayout property returns the PageLayout object for the specified Diagram object. The PageLayout object controls numerous aspects of page formatting for printing and displaying a diagram.

Example The following example retrieves the PageLayout object from the PageLayout property, and uses it to change the width of the page.

```
' Dimension the variables
Dim igxPageLayout As PageLayout
' Set the igxPageLayout variable to the PageLayout object
Set igxPageLayout = ActiveDiagram.PageLayout
MsgBox "Click OK to reduce the width of the page."
igxPageLayout.PageWidth = igxPageLayout.PageWidth / 2
MsgBox "Click OK to restore the width."
igxPageLayout.PageWidth = igxPageLayout.PageWidth * 2
MsgBox "Click OK to continue."
```

See Also PageLayout object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

PageLayoutChange Event

Syntax **Private Sub** *Diagram*_**PageLayoutChange**(ByVal *ClassID* As String)

Description The PageLayoutChange event occurs when a PageLayout property of the specified Diagram object has been modified. The event allows you to perform some set of actions in response to a change in the diagram's page layout.

The *ClassID* parameter contains the Class Identifier of the View object whose page layout changed.

Example The following example opens two diagrams, and copies the page layout of one diagram to another. When the page layout of one diagram is changed, the event copies the page layout back to the other diagram, so that they remain synchronized.

```
' Dimension a diagram variable that listens to events
Private WithEvents igxDiagram1 As Diagram

Private Sub Main()
    ' Dimension the variables
   Dim igxApp As Application
   Dim igxDiagram As Diagram
   Dim igxPageLayout As PageLayout
   ' Set the ixappApp variable to the current Application object
   Set igxApp = Application.Application
   ' Set the Diagram variables to the new Diagram objects
   Set igxDiagram1 = igxApp.ActiveDiagram
   ' Set the igxPageLayout variable to the PageLayout object
   Set igxPageLayout = igxDiagram1.PageLayout
   MsgBox "Click OK to reduce the width of the page."
   ' Change the width of the page
   igxPageLayout.PageWidth = igxPageLayout.PageWidth / 2
   MsgBox "Click OK to restore the width."
   igxPageLayout.PageWidth = igxPageLayout.PageWidth * 2
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagram1_PageLayoutChange(ByVal ClassID As String)
     MsgBox "The page layout has been modified."
End Sub
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Pages Property

Syntax *Diagram*.**Pages**

Data Type Pages collection object (read-only, See Object Properties)

Description The Pages property returns the Pages collection for the specified Diagram object. For each page of the diagram that is occupied by diagram objects, there is a Page object in the Pages collection.

Example The following example creates shapes on multiple pages, and then displays the size of the page grid that the shapes occupy.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
' Add the shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 9)
' Add another shape, but on the next page to the right
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 9, 1440)
' Connect the shapes with a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Display a message indicating the number of pages
MsgBox "The added shapes occupy a grid of " _
& ActiveDiagram.Pages.Count & " pages."
```

See Also Page object

Pages object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Paste Method

Syntax *Diagram*.**Paste** (*X* As Long, *Y* As Long) As Boolean

Description The Paste method pastes the contents of the clipboard into the specified diagram. Use the Copy and Cut methods to add diagram objects to the clipboard for pasting. The Paste method pastes non-DiagramObject items as well. For instance, if the clipboard contains text copied from another application, the text is pasted into the diagram as a TextGraphic.

The X and Y arguments specify the position coordinates to paste the objects. The units are in twips (1440 twips = 1 inch). The position coordinates are based on the upper left corner of the diagram, and the upper left corner of the collection of objects being pasted.

Example The following example adds two objects to the diagram. It then selects and copies the objects. Then it pastes the objects back into the diagram at a different location.

```
' Add two shapes to the diagram
ActiveDiagram.DiagramObjects.AddShape 1440, 1440
ActiveDiagram.DiagramObjects.AddShape 1440 * 3, 1440
' Select all the shapes in the diagram
ActiveDiagram.Selection.AddAll ixObjectShape
' Copy the shapes
MsgBox "Click OK to copy and paste the objects to a new location."
ActiveDiagram.Copy
' Paste the shapes
ActiveDiagram.Paste 1440, 1440 * 3
MsgBox "Click OK to continue."
```

See Also Copy method

Cut method

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

PasteDiagram Method

Topic Under Construction!!!

Syntax *Diagram*.**PasteDiagram**

Description The PasteDiagram method pastes

Example The following example

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

PasteLink Method

Syntax *Diagram*.**PasteLink**(*X* As Long, *Y* As Long) As Boolean

Description The PasteLink method pastes an item from the clipboard into the specified diagram as an OLE link. This is useful for pasting OLE objects, such as a Word documents, into the diagram as a link, instead of embedding the object.

The difference between a *linked* OLE object and an *embedded* object is that the linked object is dependent on it's parent application. If the parent application makes changes to the OLE object, those changes appear in the linked object in iGrafx Professional.

The *X* and *Y* arguments specify the location at which to paste the top, left corner of the OLE object.

Example The following example pastes a Word document onto the diagram as a link. This example requires a Word document on disk. The example expects this file to be called "C:\Sample.doc". Otherwise, you can substitute your own document (it must support OLE).

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxWordDoc As OleObject
' Embed a Word document
Set igxWordDoc = ActiveDiagram.DiagramObjects.AddOleObject _
    ("C:\Sample.doc", 5000, 4000)
' Select the Ole object
ActiveDiagram.Selection.Add igxWordDoc.DiagramObject
' Cut it to the clipboard
MsgBox "Click OK to cut the Ole Word document"
ActiveDiagram.Cut
' Re-paste it as a link
MsgBox "Click OK to re-paste it as a link."
ActiveDiagram.PasteLink 5000, 4000
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

PasteSpecial Method

Syntax *Diagram*.**PasteSpecial** (*X* As Long, *Y* As Long, [*Format* As IxPasteFormat = ixNative], [*AsIcon* As Boolean = False]) As Boolean

Description The PasteSpecial method pastes the contents of the clipboard into the diagram. PasteSpecial has the added fuctionality (compared to the Paste method) to paste special types of clipboard items, such as metafiles, DIB files, bitmaps, and OLEClientLinks. For instance, if the clipboard contains a bitmap, it can be pasted as a metafile, a bitmap, or a Device Independent Bitmap.

The *X* and *Y* arguments specify the coordinate location to paste the items. The units are in twips (1440 twips = 1 inch). The position is based on the upper left corner of the diagram, and the upper left corner of the items being pasted.

The *Format* argument specifies the format of the item to paste. The IxPasteFormat constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixNative
5	ixMetafile
6	ixDeviceIndependentBitmap
7	ixBitmap
8	ixOLEClientLink

The *AsIcon* argument specifies whether the pasted item is displayed as an icon or as the actual item. If *AsIcon* is set to True, an icon is added to the diagram instead of the actual item, and the icon is linked to the actual item. If *AsIcon* is set to False, the item is pasted into the diagram normally, with its normal appearance.

Example The following example pastes an image onto the active diagram as a metafile.

To try this example, first create a bitmap and copy it to the clipboard. Open Paint (included with Microsoft Windows). Use the selection tool to select a small area, and select Edit ->Copy. Then return to iGrafx Professional and run this example.

```
' Dimension the variables
Dim igxApp As Application
Dim igxDiagram As Diagram
' Set the ixappApp variable to the current Application object
Set igxApp = Application.Application
' Set the Diagram variables to the new Diagram objects
Set igxDiagram = igxApp.ActiveDiagram
' Pastes a bitmap from the clipboard into the diagram
igxDiagram.PasteSpecial 1440 * 3, 1440 * 3, ixMetafile
```

See Also Copy method

Cut method

Paste method

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

PermanentDiagram Property

Syntax *Diagram*.**PermanentDiagram**

Data Type Diagram object (read-only, See Object Properties)

Description The PermanentDiagram property returns a Diagram object. The purpose of this property is to provide a means of holding on to the object an AnyControl is pointing at after an event is over.

The AnyControl objects are special VBA controls that are only valid during an event; these objects dynamically point at the "active" object that is triggering the event. The PermanentDiagram property is used to “grab” the specific object the AnyControl is pointing at so that it can be used (or accessed) once the event is over.

As an example, consider the following event procedure written for the AnyDiagram_BeforeClick event.

```
Private Sub AnyDiagram_BeforeClick()
    Set MyDiagram = AnyDiagram
End Sub
```

If the variable MyDiagram is a global variable of type Diagram, then within the BeforeClick event you can set MyDiagram to the Diagramobject that is currently active. However, if you try to use MyDiagram after the event is over, it returns an error because an event is not in progress. Since you set MyDiagram to the AnyControl, your variable is pointing at the AnyControl that is dynamically pointing at the active object, which is Nothing outside of an event.

If your intent is to hold on to the specific diagram that the AnyDiagram control is pointing at inside the event, then you need to use the PermanentDiagram property. This property gives you a Diagram object that is valid after the event is over (outside of the event). The change to your code is as follows (MyDiagram is a global variable of type Diagram):

```
Private Sub AnyDiagram_BeforeClick()
    Set MyDiagram = AnyDiagram.PermanentDiagram
End Sub
```

Example The following example has a block of module variables, two subroutines, and one event. In the first subroutine, one extra diagram is added to the document, and the AnyDiagram object is set. The user is then instructed to return to the document and activate diagrams, which triggers the Activate event. The second subroutine indicates the last diagram that was activated, which was set in the event by the PermanentDiagram property.

```
' Dimension module level variables
Private WithEvents igxDiagram1 As Diagram
Private WithEvents igxDiagram2 As Diagram
Private WithEvents MyAnyDiagram As Diagram
' No need for WithEvents here because this variable
' inherits WithEvents when set later
Private igxDiagram3 As Diagram

Private Sub RunThisFirst()
   ' Set MyAnyDiagram to listen for events on any diagram
   Set MyAnyDiagram = ActiveDocument.AnyControls.AnyDiagram
   ' Set variable to the active diagram
   Set igxDiagram1 = ActiveDiagram
   ' Set variable to a new diagram
   Set igxDiagram2 = ActiveDocument.Diagrams.Add("Diagram B")
```

```
   MsgBox "Diagrams set.  Go to the diagram and activate a diagram."
End Sub

Private Sub RunThisSecond()
   ' Display which diagram was last activated
   MsgBox "The last activated diagram was " & igxDiagram3.Name
End Sub

Private Sub MyAnyDiagram_Activate()
   ' Retrieve the diagram form which the event originated
   Set igxDiagram3 = MyAnyDiagram.PermanentDiagram
End Sub
```

See Also iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

PrintDiagram Method

Syntax *Diagram*.**PrintDiagram**

Description The PrintDiagram method prints the specified diagram on the system printer.

Example The following example prints the active diagram.

```
' Dimension the variables
Dim igxApp As Application
Dim igxDiagram As Diagram
' Set the ixappApp variable to the current Application object
Set igxApp = Application.Application
' Set the Diagram variables to the new Diagram objects
Set igxDiagram = igxApp.ActiveDiagram
' Prints the active diagram
If MsgBox("Print the diagram?", vbYesNo + vbExclamation) _
= vbYes Then
    igxDiagram.PrintDiagram
End If
```

See Also AfterPrint event

BeforePrint event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

PropertyChange Event

Syntax **Private Sub** *Diagram*_**PropertyChange**(*Property* As Property)

Description The PropertyChange event occurs when a Property in the specified diagram is changed. The Property parameter contains the Property object that was changed.

A Property object is a custom property added to the diagram by the programmer. A Property object is a member of the PropertyList collection, which in turn is a member of the PropertyLists collection of the diagram. You can add Property objects to a diagram using the PropertyLists object.

Example The following example adds a property, called "Modified", to the active diagram. The property is set to the current date and time. When the property value is set, the PropertyChange event is fired.

```
Sub Main()
    Dim igxDiagram As Diagram
   Dim igxPropertyLists As PropertyLists
   Dim igxPropertyList As PropertyList
   Dim igxProperty As Property
   ' Set the active diagram variable
   Set igxDiagram = ActiveDiagram
   ' Add/Set a propertylist
   Set igxPropertyList = igxDiagram.PropertyLists.Add("Attributes")
   ' Add/Set a Property
   Set igxProperty = igxPropertyList.Add("Modified")
   ' Change the property to the current date and time
   igxProperty.Value = Now
   MsgBox "Click Ok to change the Modified property of this diagram."
   ' Again change the property to the current date and time
   igxProperty.Value = Now
End Sub

Private Sub Diagram_PropertyChange(ByVal Property As Property)
   ' Display the changed property
   MsgBox "The diagram's " & Property.Name & _
       " property was just changed to " & Property.Value
End Sub
```

See Also PropertyLists property

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

PropertyLists Property

Syntax *Diagram*.**PropertyLists**

Data Type PropertyLists collection object (read-only, See Object Properties)

Description The PropertyLists property returns the PropertyLists collection for the specified Diagram object. The PropertyLists object is a collection of PropertyList objects, each of which can contain programmer-defined Property objects.

Example The following example adds a property, called "Modified", to the active diagram. The property is set to the current date and time. When the property value is set, the PropertyChange event is fired.

```
Sub Main()
    Dim igxDiagram As Diagram
    Dim igxPropertyLists As PropertyLists
    Dim igxPropertyList As PropertyList
    Dim igxProperty As Property
    ' Set the active diagram variable
    Set igxDiagram = ActiveDiagram
    ' Add/Set a propertylist
    Set igxPropertyList = igxDiagram.PropertyLists.Add("Attributes")
    ' Add/Set a Property
    Set igxProperty = igxPropertyList.Add("Modified")
    ' Change the property to the current date and time
    igxProperty.Value = Now
    MsgBox "Click Ok to change the Modified property of this diagram."
    ' Again change the property to the current date and time
    igxProperty.Value = Now
End Sub

Private Sub Diagram_PropertyChange(ByVal Property As Property)
    ' Display the changed property
    MsgBox "The diagram's " & Property.Name & _
        " property was just changed to " & Property.Value
End Sub
```

See Also PropertyList object

PropertyLists object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Refresh Method

Syntax *Diagram*.**Refresh** As Boolean

Description The Refresh method causes iGrafx Professional to be completely redraw the diagram. If unexpected screen artifacts appear on the diagram, use the Refresh method to repaint the diagram from scratch.

Example The following example adds a new layer to the diagarm, and refreshes the display.

```
' Dimension the variables
Dim igxDiagram As Diagram
' Set the active diagram variable
Set igxDiagram = ActiveDiagram
' Add a layer to the diagram
MsgBox "Click Ok to add a layer to the diagram."
igxDiagram.Layers.Add ("LayerB")
' Refresh the display
MsgBox "Click OK to refresh the display."
igxDiagram.Refresh
MsgBox "Click OK to continue."
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Rename Event

Syntax **Private Sub** *Diagram*_**Rename**(ByVal *OldName* As String)

Description The Rename event occurs when the specified diagram is renamed. A diagram can be renamed by changing the value of the Diagram.Name property. The *OldName* parameter contains the name of the diagram before it was renamed.

Example The following example displays a message when the diagram is renamed.

```
' Dimension a diagram variable that hears events
Private WithEvents MyAnyDiagram As Diagram

Sub Main()
   Dim igxDiagram As Diagram
   ' Set the active diagram variable
   Set igxDiagram = ActiveDiagram
   Set MyAnyDiagram = ActiveDocument.AnyControls.AnyDiagram
   ' Rename the diagram
   MsgBox "Click Ok to rename the diagram to Diagram X"
   igxDiagram.Name = "Diagram X"
End Sub

Private Sub MyAnyDiagram_Rename(ByVal OldName As String)
   ' Display the result
   MsgBox OldName & " has been renamed to " & MyAnyDiagram.Name
End Sub
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

ReplaceText Method

Syntax *Diagram*.**ReplaceText**(*FindText* As String, *ReplaceText* As String, [*LookIn* As IxLookIn = ixLookInChart], [*MatchCase* As Boolean = False], [*MatchWholeWord* As Boolean = False]) As Long

Description The ReplaceText method performs a search-and-replace of any text elements in the diagram. The method returns a Long value that contains the number of replacements made.

The *FindText* argument specifies the text to search for.

The *ReplaceText* argument specifies the text to substitute.

The *MatchCase* argument specifies whether upper and lower case is relevant in the search. If set to True, case is relevant, and the case must match to be considered a valid match. If set to False, case is not relevant and is ignored.

The *MatchWholeWord* argument specifies whether the *FindText* string is considered a complete word, or a sub-string within words. If set to True, the search does not look for matching strings inside of words—only complete words are considered a valid match and replaced. If set to False, the search looks for the *FindText* string as a sub-string inside of words, and replaces the sub-string (which may or may not be a complete word).

The *LookIn* argument specifies what text diagram elements are searched. You can have the search look in diagrams only, custom data definitions only, notes only, or search through all text elements in the diagram. The IxLookin constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixLookInChart
1	ixLookInCustomData
2	ixLookInNotes
3	ixLookInAll

Example The following example searches for, and replaces, text in the active diagram.

```
' Dimension the variables
Dim igxShape As Shape
Dim Result As Long
' Add a shape in the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Autogrow allows the shape to expand to fit the text
igxShape.AutoGrow = True
' Add a line of text to the shape
igxShape.Text = "My activity takes my time and my resources."
' Pause for user
MsgBox "Click OK to change the word MY to YOUR."
' Replace the word MY with YOUR everywhere in the diagram
Result = igxDiagram.ReplaceText("my", "your")
' Pause for user
MsgBox Result & " replacements were made."
```

See Also CheckSpelling method

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Save Event

Syntax **Private Sub** *Diagram*_**Save**()

Description The Save event occurs when the specified Diagram object is saved. The events allows the developer to perform some set of actions when the Save event is handled by iGrafx Professional. A diagram is saved when it's parent document is saved.

Example The following example stores and displays the last date and time that the diagram was saved.

```
' Dimension a module variable for storing the date
Private LastSaved As Date

Private Sub Main()
   ' Save the document, therefore the diagram
   ActiveDocument.SaveDocumentAs "C:\test.igx"
End Sub

Private Sub Diagram_Save()
   ' Store the data and time
   LastSaved = Now
   MsgBox "This diagram was last saved " & LastSaved
End Sub
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

SaveAsWebPage Event

Syntax **Private Sub** *Diagram*_**SaveAsWebPage**

Description The SaveAsWebPage event occurs before the specified diagram is saved to disk as a web page. The event happens when the SaveAsWebPage method is invoked in Visual Basic, or when the user goes to the iGrafx Professional File menu and chooses File->Save As Web Page. Use this event to perform any custom page setup changes needed before exporting a diagram as a web page.

Example The following example uses the AnyDiagram object to listen to all SaveAsWebPage events. If a diagram is about to be saved as a web page, the event automatically changes the diagram's page title mode to "PerDiagram".

```
Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxDiagRange As DiagramRange
    ' Create 3 shapes and assign the shape variables to
    ' the 3 Shape objects
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 4, 1440)
    ' Create a diagram range object
    Set igxDiagRange = ActiveDocument.MakeDiagramRange
    igxDiagRange.Add ActiveDiagram
    ' Save the document as a web page
    ActiveDocument.SaveAsWebPage igxDiagRange, , _
        "C:\Program Files\iGrafx\Pro\8.0"
End Sub

Private Sub AnyDiagram_SaveAsWebPage()
    MsgBox "Page title mode has been changed to PerDiagram." & _
        Chr(13) & "Click OK to proceed with saving HTML page."
    AnyDiagram.PageLayout.PageTitleMode = ixPerDiagram
End Sub
```

See Also AfterSaveAsWebPage event

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Selection Property

Syntax *Diagram*.**Selection**

Data Type ObjectRange object (read-only, See Object Properties)

Description The Selection property returns an ObjectRange object for the specified Diagram object. The ObjectRange object provides properties and methods for selecting diagram objects, which can then be copied or cut to the clipboard.

Example The following example adds two objects to the diagram. It then selects and copies the objects. Then it pastes the objects back into the diagram at a different location.

```
' Add two shapes to the diagram
ActiveDiagram.DiagramObjects.AddShape 1440, 1440
ActiveDiagram.DiagramObjects.AddShape 1440 * 3, 1440
' Select all the shapes in the diagram
ActiveDiagram.Selection.AddAll ixObjectShape
' Copy the shapes
MsgBox "Click OK to copy and paste the objects to a new location."
ActiveDiagram.Copy
' Paste the shapes
ActiveDiagram.Paste 1440, 1440 * 3
MsgBox "Click OK to continue."
```

See Also ObjectRange object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

SelectionChange Event

Syntax **Private Sub** *Diagram*_**SelectionChange**(*Selection* As ObjectRange)

Description The SelectionChange event occurs when the “selection” for the Diagram object is changed either by the user or programmatically. The Selection property is an ObjectRange object that contains all the selected objects in the diagram. Any change to the Selection property’s contents fires the SelectionChange event. The Selection parameter contains the diagram’s Selection property (an ObjectRange object).

Example The following example sets up a SelectionChange event. Copy this code into a diagram's code class window. Then go to the diagram and try selecting different diagram objects.

```
Private Sub MyAnyDiagram_SelectionChange(ByVal Selection As ObjectRange)
    ' Dimension a string variable
   Dim sString As String
   ' Collect the name of all the objects in the selection
   For Index = 1 To Selection.Count
       sString = sString & Selection.Item(Index).ObjectName & Chr(13)
   Next Index
   ' Display the result
   MsgBox "The new selection contains these objects:" & Chr(13) _
       & Chr(13) & sString
End Sub
```

See Also Selection property

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

StartPointNames Property

Syntax *Diagram*.**StartPointNames**

Data Type StartPointNames object (read-only, See Object Properties)

Description The StartPointNames property returns the StartPointNames object for the specified diagram. The StartPointNames object is a collection of strings that contains the names of all the shapes that are starting points for entities within the diagram.

Example The following example displays the start point names in the diagram.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxStartPointNames As StartPointNames
' Add two shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
' Set start point names for the shapes
igxShape1.StartPointName = "Start Point A"
igxShape2.StartPointName = "Start Point B"
' Get the StartPointNames object for the diagram
Set igxStartPointNames = ActiveDiagram.StartPointNames
' Collect the start point names into a string
For Index = 1 To igxStartPointNames.Count
   sString = sString & igxStartPointNames.Item(Index) & Chr(13)
Next Index
' Display the result
MsgBox "The diagram contains these Start Point Names:" & Chr(13) _
    & Chr(13) & sString
```

See Also StartPointNames object

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

UpdateFields Method

Topic Under Construction!!!

Syntax *Diagram*.**UpdateFields**

Description The UpdateFields method

Example The following example

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

UserEvent Event

Syntax **Private Sub** *Diagram*_**UserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The UserEvent event provides a means of implementing your own custom events. Your custom events can then be triggered with the FireUserEvent method, which fires the specified "UserEvent" on the document. You can use this functionality to send messages to any objects listening to document-level events.

You must pick an event identifier string to use for your event. You might choose to use something like your company name followed by the event name. You should choose a name that won't conflict with names picked by other developers.

You can pass one parameter to the event. This parameter is a Variant, so one logical choice is to pass a class.

You then write code in a UserEvent handler to perform some actions when your event fires. This code should be of the form:

```
If EventIdentifier = "<<Your identifier string>>" Then
   << Write your code here >>
End If
```

Example The following example defines a new user event called "ShowUsers". The *Parameter* that gets passed is a class, which as one property called Count. The event handler displays the passed parameter's Count property.

The following code creates a simple class with one property. Create a new class below a diagram project called Class1 and copy this code into it.

```
' Class1
' It contains one property, read only
Public Property Get Count() As Long
   Count = 25
End Property
```

The following code is the main program. Copy this, and the UserEvent subroutine, into the diagram project code window

```
' Run this subroutine to test the event
Public Sub Main()
   ' Fire the UserEvent
   Diagram.FireUserEvent "ShowUsers", New Class1
End Sub

' This event handler runs every time any FireUserEvent method
' is used in the system
Private Sub Diagram_UserEvent(ByVal EventIdentifier As String, ByVal Parameter
As Variant)
   ' Check if the Identifier string is the one we want
   If EventIdentifier = "ShowUsers" Then
       ' Redirect to Class1
       MsgBox "The number of users is " & Parameter.Count
   End If
End Sub
```

See Also FireUserEvent method

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

VBAName Property

Syntax *Diagram*.**VBAName**

Data Type String (read/write)

Description The VBAName property is a string value that specifies the Visual Basic name of a Diagram object. Only letters and numbers can be used. No spaces, punctuation marks, or other special characters can be used in the VBAName property.

Example The following example displays the VBAName of the active diagram, then changes the name, and displays the name again.

```
' Dimension the variables
Dim igxDiagram As Diagram
' Get the ActiveDocument object
Set igxDiagram = Application.ActiveDiagram
' Display the current VBAName
MsgBox "The VBA Name of the diagram is " & igxDiagram.VBAName
igxDiagram.VBAName = "MyRenamedDiagram"
MsgBox "The VBA Name of the diagram is " & igxDiagram.VBAName
```

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Views Property

Syntax *Diagram*.**Views**

Data Type Views collection object (read-only, See Object Properties)

Description The Views property returns the Views collection for the specified Diagram object. The Views object is a collection that contains all of the views that have been created for the diagram.

Example The following example gets the diagram's Views collection, and displays the zoom percentage of the current view.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxViews As Views
' Get the ActiveDiagram object
Set igxDiagram = ActiveDiagram
' Get the Views object
Set igxViews = igxDiagram.Views
' Display the result
MsgBox "The Zoom Percentage of the current diagram view is " & _
    igxViews.Item(1).DiagramView.ZoomPercentage
```

See Also View object

Views object

iGrafx API Object Hierarchy

{button Diagram object,JI(`igrafxrf.HLP',`Diagram_Object')}

Diagrams Object

The Diagrams object is a collection of individual Diagram objects. A Diagrams collection is only associated with and accessible from the Document object. Its purpose is to store and provide access to the individual Diagram objects that have been created in a document.

The Diagrams object provides the following functionality:

- The ability to access any Diagram objects that have created in a document.
- The ability to determine how many Diagram objects are in the collection.
- The ability to have iGrafx Professional suggest a name for the next Diagram object that is created in a document.
- The ability to add a new Diagram object to a document.

Properties, Methods, and Events

All of the properties, methods, and events for the Diagrams object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	AddFromTemplate	
NextSuggestedName	AddOfType	
NextSuggestedNameOfType	Item	
Parent		

Add Method

Syntax *Diagrams*.**Add** ([*DiagramName* As String = "Untitled"], [*Show* As Boolean = True]) As Diagram

Description The Add method adds a new diagram to a document. The *DiagramName* argument specifies the name of the diagram. Each diagram in a document must have a unique DiagramName. The *Show* argument specifies whether the new diagram is displayed in the user interface after it's created. If set to False, the diagram is added as a diagram project, but is not displayed in the user interface.

Example The following example adds a new diagram to the active document.

```
' Dimension the variables
Dim igxDiagram As Diagram
MsgBox "Click OK to create a new diagram."
' Add a diagram to the document, set a diagram variable,
' and show the diagram in the user interface
Set igxDiagram = ActiveDocument.Diagrams.Add("Diagram D", True)
MsgBox "Click OK to continue."
```

See Also AddFromTemplate method

AddOfType method

{button Diagrams object,JI(`igrafxrf.HLP',`Diagrams_Object')}

AddFromTemplate Method

Syntax *Diagrams*.**AddFromTemplate**([*DiagramName* As String = "Untitled"], *TemplateName* As String, *DiagramInTemplateName* As String, [*Show* As Boolean = True]) As Diagram

Description The AddFromTemplate method adds a new diagram to a document that is derived from a template. The AddFromTemplate method returns the Diagram object that was added.

The *DiagramName* argument is the name the programmer chooses for the new diagram being added.

The *TemplateName* argument specifies the name of a template file on disk. This argument can include a full path name to the template file, or can just specify a file name and use the default path specified by the Templates.DefaultTemplatePath property.

The *DiagramInTemplateName* argument specifies the name of a diagram within the template from which to derive the new diagram.

The *Show* argument specifies whether the new diagram appears in the user interface. If set to False, the new diagram is added as a diagram project, but the diagram does not appear in the user interface.

Example The following example adds a new diagram to the document, derived from the Cascade Template.

```
' Dimension the variables
Dim igxTemplates As Templates
Dim igxDiagrams As Diagrams
' Set the igxTemplates variable to the Templates collection
Set igxTemplates = Application.Templates
' Set the Diagrams object
Set igxDiagrams = ActiveDocument.Diagrams
' Point the template path to the iGrid templates
igxTemplates.DefaultTemplatePath = _
   "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid"
' Create a new diagram from the Cascade template
MsgBox "Click OK to create a new diagram from the Cascade Template"
igxDiagrams.AddFromTemplate "MyCascadeDiagram", _
   "Cascade.igt", "Cascade", True
MsgBox "Document created. Click OK to continue."
```

See Also Add method

AddOfType method

{button Diagrams object,JI(`igrafxrf.HLP',`Diagrams_Object')}

AddOfType Method

Syntax *Diagrams*.**AddOfType**([*DiagramName* As String = "Untitled"], *DiagramType* As DiagramType, [*Show* As Boolean = True]) As Diagram

Description The AddOfType method adds a new diagram of a particular type to the document. The AddOfType method returns the diagram object that was added.

The *DiagramName* argument is the name the programmer chooses for the new diagram being added.

The DiagramType argument is a DiagramType object, which determines the type of diagram created. iGrafx Professional has two built-in diagram types: Basic and Process.

The *Show* argument specifies whether the new diagram appears in the user interface. If set to False, the new diagram is added as a diagram project, but the diagram does not appear in the user interface.

Example The following example adds a new "Process" type diagram to the active document.

```
' Dimension the variables
Dim igxDiagrams As Diagrams
Dim igxDiagramType As DiagramType
' Set the Diagrams object
Set igxDiagrams = ActiveDocument.Diagrams
' Set the DiagramType variable,  Item(1) type "Process"
Set igxDiagramType = DiagramTypes.Item(1)
' Create a new "Process" type diagram
MsgBox "Click OK to create a Process type diagram."
igxDiagrams.AddOfType "Process Diagram B", igxDiagramType, True
MsgBox "Document created. Click OK to continue."
```

See Also Add method

AddFromTemplate method

{button Diagrams object,JI(`igrafxrf.HLP',`Diagrams_Object')}

Item Method

Syntax *Diagrams*.**Item**(*Index* As Integer) As Diagram

Description The Item method returns the Diagram object at the specified *Index* from the Diagrams collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Diagram. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example displays the names of all the Diagram objects in the Diagrams collection, using the Item method.

```
' Collect all the diagram names into a string
For Index = 1 To ActiveDocument.Diagrams.Count
    sString = sString & ActiveDocument.Diagrams _
        .Item(Index).Name & Chr(13)
Next Index
' Display the result
MsgBox "The document contains these diagrams:" & Chr(13) _
    & Chr(13) & sString
```

{button Diagrams object,JI(`igrafxrf.HLP',`Diagrams_Object')}

NextSuggestedName Property

Syntax *Diagrams*.**NextSuggestedName**

Data Type String (read-only)

Description The NextSuggestedName property returns a string containing the next suggested name to use for a new diagram. This property provides a convenient way of naming a new diagram when adding a new diagram to a document. When adding new diagrams to a document, each diagram must have a unique name. The NextSuggestedName property guarantees that the name used to create the new document is unique.

Example The following example creates a new document using the NextSuggestedName property.

```
' Add a new diagram to the document. For the DiagramName argument,
' use the NextSuggestedName property
ActiveDocument.Diagrams.Add ActiveDocument.Diagrams _
    .NextSuggestedName, True
```

See Also Add method

NextSuggestedNameOfType property

{button Diagrams object,JI(`igrafxrf.HLP',`Diagrams_Object')}

NextSuggestedNameOfType Property

Syntax *Diagrams*.**NextSuggestedNameOfType**(*DiagramType* As DiagramType) As String

Data Type String (read-only)

Description The NextSuggestedNameOfType property returns a string containing the next suggested name of a specific diagram type to use for a new diagram. This property provides a convenient way of naming a new diagram when adding a new diagram to the document.

When adding new diagrams to a document, each diagram must have a unique name. The NextSuggestedNameOfType property guarantees that the name used to create the new document is unique. This property allows the programmer to organize diagrams based on diagram type. When using a different naming pattern for Process and Basic diagrams, the NextSuggestedNameOfType property preserves the two naming patterns.

Example The following example adds a new "Process" type diagram to the document, and names the diagram according to the NextSuggestedNameOfType property.

```
' Dimension the variables
Dim igxDiagrams As Diagrams
Dim igxDiagramType As DiagramType
' Set the Diagrams object
Set igxDiagrams = ActiveDocument.Diagrams
' Set the DiagramType variable, Item(1) type is "Process"
Set igxDiagramType = DiagramTypes.Item(1)
' Create a new "Process" type diagram
MsgBox "Click OK to create a Process type diagram."
' Create a new "Process" type diagram
igxDiagrams.AddOfType ActiveDocument.Diagrams _
    .NextSuggestedNameOfType(igxDiagramType), _
   igxDiagramType, True
MsgBox "Process diagram created. Click OK to continue."
```

See Also Add method

AddOfType method

NextSuggestedName property

{button Diagrams object,JI(`igrafxrf.HLP',`Diagrams_Object')}

DiagramRange Object

A DiagramRange object is a collection of Diagram objects. If you are working with multiple diagrams within a document, one or more of the diagrams can be collected into a DiagramRange, and then manipulated or acted upomn in some fashion.

The DiagramRange object provides the following functionality:

- The ability to access any Diagram objects that are in the DiagramRange collection.
- The ability to determine how many Diagram objects are in the DiagramRange collection.
- The ability to add individual Diagrams to the range, or add to the range based on the contents of another DiagramRange object.
- The ability to remove Diagram objects from the range, either individually or all of them at once, or based on the contents of another DiagramRange object.

Properties, Methods, and Events

All of the properties, methods, and events for the DiagramRange object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	AddRange	
Parent	Item	
	Remove	
	RemoveAll	
	RemoveRange	

Add Method

Syntax *DiagramRange*.**Add**(*Diagram* As Diagram)

Description The Add method adds a diagram to the DiagramRange. The *Diagram* argument specifies the Diagram object to add.

Example The following example creates a DiagramRange object, and adds a diagram to it.

```
' Dimension the variables
Dim igxDiagramRange As DiagramRange
Dim igxDiagram As Diagram
' Make a new DiagramRange
Set igxDiagramRange = ActiveDocument.MakeDiagramRange
' Add a new diagram to the document
Set igxDiagram = ActiveDocument.Diagrams.Add(ActiveDocument _
    .Diagrams.NextSuggestedName)
' Add the diagram to the DiagramRange
igxDiagramRange.Add igxDiagram
MsgBox "Diagram added."
```

See Also AddRange method

{button DiagramRange object,JI(`igrafxrf.HLP',`DiagramRange_Object')}

AddRange Method

Syntax *DiagramRange*.**AddRange**(*Range* As DiagramRange)

Description The AddRange method adds the contents of the DiagramRange object specified by the *Range* argument to another DiagramRange object.

Example The following example creates three diagrams, and two DiagramRange objects. It assigns diagrams to each range, and then adds one DiagramRange to another.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxDiagram3 As Diagram
Dim igxDiagramRange1 As DiagramRange
Dim igxDiagramRange2 As DiagramRange
Dim sString As String
' Create three new diagrams
Set igxDiagram1 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram2 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram3 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
' Create two DiagramRange objects
Set igxDiagramRange1 = ActiveDocument.MakeDiagramRange
Set igxDiagramRange2 = ActiveDocument.MakeDiagramRange
' Add one diagram to the first DiagramRange
igxDiagramRange1.Add igxDiagram1
' Add two diagrams to the second DiagramRange
igxDiagramRange2.Add igxDiagram2
igxDiagramRange2.Add igxDiagram3
' Add the second DiagramRange to the first
igxDiagramRange1.AddRange igxDiagramRange2
' Collect the memeber names of the DiagramRange into a string
For Index = 1 To igxDiagramRange1.Count
   sString = sString & igxDiagramRange1.Item(Index).Name & Chr(13)
Next Index
' Display the result
MsgBox "The DiagramRange contains these diagrams:" & Chr(13) _
   & Chr(13) & sString
```

See Also Add method

RemoveRange method

{button DiagramRange object,JI(`igrafxrf.HLP',`DiagramRange_Object')}

Item Method

Syntax *DiagramRange*.**Item**(*Index* As Integer) As Diagram

Description The Item method returns the Diagram object at the specified *Index* from the DiagramRange collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Diagram. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example creates three diagrams and adds them to a DiagramRange. It then displays the names of the diagrams contained in the range, using the Item method.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxDiagram3 As Diagram
Dim igxDiagramRange1 As DiagramRange
Dim sString As String
' Create three new diagrams
Set igxDiagram1 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram2 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram3 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
' Create a DiagramRange object
Set igxDiagramRange1 = ActiveDocument.MakeDiagramRange
' Add the diagrams to the DiagramRange
igxDiagramRange1.Add igxDiagram1
igxDiagramRange1.Add igxDiagram2
igxDiagramRange1.Add igxDiagram3
' Collect the memeber names of the DiagramRange into a string
For Index = 1 To igxDiagramRange1.Count
   sString = sString & igxDiagramRange1.Item(Index).Name & Chr(13)
Next Index
' Display the result
MsgBox "The DiagramRange contains these diagrams:" & Chr(13) _
   & Chr(13) & sString
```

{button DiagramRange object,JI(`igrafxrf.HLP',`DiagramRange_Object')}

Remove Method

Syntax *DiagramRange*.**Remove**(*Diagram* As Diagram)

Description The Remove method removes a diagram from the DiagramRange collection. The *Diagram* argument specifies a valid Diagram object contained in the range. An error is returned if the *Diagram* argument does not specify a valid Diagram object.

Example The following example creates a DiagramRange and adds three diagrams to it. It then removes one of the diagrams from the range, and displays the result.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxDiagram3 As Diagram
Dim igxDiagramRange1 As DiagramRange
Dim sString As String
' Create three new diagrams
Set igxDiagram1 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram2 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram3 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
' Create a DiagramRange object
Set igxDiagramRange1 = ActiveDocument.MakeDiagramRange
' Add the diagrams to the DiagramRange
MsgBox "Click OK to add three diagrams to the DiagramRange"
igxDiagramRange1.Add igxDiagram1
igxDiagramRange1.Add igxDiagram2
igxDiagramRange1.Add igxDiagram3
' Remove one of the diagrams
MsgBox "Now click OK to remove one of the diagrams from the range."
igxDiagramRange1.Remove igxDiagram2
' Collect the memeber names of the DiagramRange into a string
For Index = 1 To igxDiagramRange1.Count
   sString = sString & igxDiagramRange1.Item(Index).Name & Chr(13)
Next Index
' Display the result
MsgBox "Now the DiagramRange contains these diagrams:" & Chr(13) _
   & Chr(13) & sString
```

See Also RemoveAll method

RemoveRange method

{button DiagramRange object,JI(`igrafxrf.HLP',`DiagramRange_Object')}

RemoveAll Method

Syntax *DiagramRange*.**RemoveAll**

Description The RemoveAll method removes all Diagram objects from the specified DiagramRange object. This method empties the DiagramRange object.

Example The following example creates three diagrams and adds them to a DiagramRange object. It then removes all the diagrams from the range, and displays the result.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxDiagram3 As Diagram
Dim igxDiagramRange1 As DiagramRange
Dim sString As String
' Create three new diagrams
Set igxDiagram1 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram2 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram3 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
' Create a DiagramRange object
Set igxDiagramRange1 = ActiveDocument.MakeDiagramRange
' Add the diagrams to the DiagramRange
MsgBox "Click OK to add three diagrams to the DiagramRange"
igxDiagramRange1.Add igxDiagram1
igxDiagramRange1.Add igxDiagram2
igxDiagramRange1.Add igxDiagram3
' Remove one of the diagrams
MsgBox "Now click OK to remove all the diagrams from the range."
igxDiagramRange1.RemoveAll
' Collect the memeber names of the DiagramRange into a string
For Index = 1 To igxDiagramRange1.Count
   sString = sString & igxDiagramRange1.Item(Index).Name & Chr(13)
Next Index
' Display the result
MsgBox "Now the DiagramRange contains these diagrams:" & Chr(13) _
   & Chr(13) & sString
```

See Also Remove method

RemoveRange method

{button DiagramRange object,JI(`igrafxrf.HLP',`DiagramRange_Object')}

RemoveRange Method

Syntax *DiagramRange*.**RemoveRange**(*Range* As DiagramRange)

Description The RemoveRange method removes Diagram objects from the specified DiagramRange object, based on the contents of another DiagramRange object. The RemoveRange method looks at the contents of the DiagramRange specified with the *Range* argument. If it contains any diagrams common to the first DiagramRange, those diagrams are removed from the first DiagramRange.

Example The following example creates three diagrams and two DiagramRange objects. It assigns diagrams to each DiagramRange. It then uses the RemoveRange method to remove diagrams from the first DiagramRange.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxDiagram3 As Diagram
Dim igxDiagramRange1 As DiagramRange
Dim igxDiagramRange2 As DiagramRange
Dim sString As String
' Create three new diagrams
Set igxDiagram1 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram2 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
Set igxDiagram3 = ActiveDocument.Diagrams.Add(ActiveDocument _
   .Diagrams.NextSuggestedName, True)
' Create two DiagramRange objects
Set igxDiagramRange1 = ActiveDocument.MakeDiagramRange
Set igxDiagramRange2 = ActiveDocument.MakeDiagramRange
' Add one diagram to the first DiagramRange
MsgBox "Click OK to add the three diagrams to the first DiagramRange."
igxDiagramRange1.Add igxDiagram1
igxDiagramRange1.Add igxDiagram2
igxDiagramRange1.Add igxDiagram3
' Add two diagrams to the second DiagramRange
igxDiagramRange2.Add igxDiagram2
' Remove diagarms from the first range based on the second
MsgBox "Click OK to remove the second range from the first."
igxDiagramRange1.RemoveRange igxDiagramRange2
' Collect the memeber names of the DiagramRange into a string
For Index = 1 To igxDiagramRange1.Count
   sString = sString & igxDiagramRange1.Item(Index).Name & Chr(13)
Next Index
' Display the result
MsgBox "The DiagramRange contains these diagrams:" & Chr(13) _
   & Chr(13) & sString
```

See Also Remove method

RemoveAll method

{button DiagramRange object,JI(`igrafxrf.HLP',`DiagramRange_Object')}

DiagramType Object

The DiagramType object defines a “type” for a diagram. For example, iGrafx Professional has two built-in diagram types: “Basic Diagram” and “Process” (these types can be seen from the FileàNew menu.

Every diagram has an associated DiagramType object. The DiagramType object contains read-only properties which provide information about the template that created it, such as whether the diagram is up-to-date with a template that may have been upgraded after initially creating the diagram. The DiagramType object also contains methods for setting which template a diagram is associated with, for updating the diagram to synchronize it with an upgraded template, and events that fire when a DiagramType is initialized or terminated.

Properties, Methods, and Events

All of the properties, methods, and events for the DiagramType object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
AnyControls	SetTemplate	CustomDataDefinitionChange
Application	UpdateFromTemplate	Initialize
ClassID		Terminate
ExternalTemplateDate		
NotifyWhenTemplateChanges		
Parent		
PluralName		
ProgID		
ShapeLibrary		
SingularName		
SynchronizedToDate		
TemplateName		
UpToDateWithTemplate		

AnyControls Property

Syntax *DiagramType*.**AnyControls**

Data Type AnyControls object (read-only, See Object Properties)

Description The AnyControls property returns an AnyControls object for the specified DiagramType object. The DiagramType.AnyControl object allows the programmer to listen to events coming from all the objects contained in the specified DiagramType. For example, if you wanted to know each time a layer was added to any Diagram of a particular type, you could listen to the AnyControls.AnyDiagram object's events. A new event would fire for that object each time a new layer was created.

Example The following example sets up a module level variable that listens to diagram events, two diagrams derived from the Cascade template, and an Activate event for the DiagramType.AnyControl object. After running the Main( ) subroutine, activating either of the Cascade diagrams fires the event.

```
' Dimension a module variable that hears diagram events
Private WithEvents MyAnyDiagram As Diagram

' Run this subroutine to set up the diagrams and the
' event diagram variable
Private Sub Main()
    ' Dimension the variables
   Dim igxTemplates As Templates
   Dim igxDiagrams As Diagrams
   Dim igxDiagram1 As Diagram
   Dim igxDiagram2 As Diagram
   Dim igxDiagramType As DiagramType
   ' Set the igxTemplates variable to the Templates collection.
   Set igxTemplates = Application.Templates
   ' Set the Diagrams object
   Set igxDiagrams = ActiveDocument.Diagrams
   ' Point the template path to the iGrid templates
   igxTemplates.DefaultTemplatePath = _
       "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid"
   ' Create 2 new diagrams from the Cascade template
   MsgBox "Click OK to add 2 new diagrams from the Cascade Template"
   Set igxDiagram1 = igxDiagrams.AddFromTemplate("MyCascadeA", _
       "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\Cascade.igt", _
       "Cascade", True)
   Set igxDiagram2 = igxDiagrams.AddFromTemplate("MyCascadeB", _
       "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\Cascade.igt", _
       "Cascade", True)
   ' Get the DiagramType object
   Set igxDiagramType = igxDiagram1.DiagramType
   ' Set the module variable
   Set MyAnyDiagram = igxDiagramType.AnyControls.AnyDiagram
   MsgBox "Now return to the interface and try activating diagrams."
End Sub


Private Sub MyAnyDiagram_Activate()
   MsgBox "A Basic Diagram has been activated."
End Sub
```

See Also AnyControls object

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

ClassID Property

Syntax *DiagramType*.**ClassID**

Data Type String (read-only)

Description The ClassID property returns the Class ID number of the DiagramType. More ClassID numbers may be added to iGrafx Professional in the future as new diagram types are implemented.

For the *Process* diagram type, a GUID (guaranteed unique ID) is returned:
"{7D81BF18-0794-11D2-9054-00C04F8EF9A2}"

For the *Basic Diagram* diagram type, the property returns all zeros:
"{00000000-0000-0000-0000-000000000000}"

Example The following example displayes the ClassID of each DiagramType object in the Application.

```
Private Sub Diagram_Activate()
   ' Display the ClassID of each DiagramType
   For Index = 1 To Application.DiagramTypes.Count
      Output Application.DiagramTypes.Item(Index).PluralName
      Output Application.DiagramTypes.Item(Index).ClassID
   Next Index
End Sub
```

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

CustomDataDefinitionChange Event

Syntax **Private Sub** *DiagramType*_**CustomDataDefinitionChange**

Description The CustomDataDefinitionChange event occurs when a custom data definition is changed in any Diagram object in any diagram of a particular type. One use of this event would be to update the data values in all diagrams of the specified “type” if the data definition is changed in any one of the diagrams.

Example The following example sets up one shape, and a CustomDataDefinition. When the user changes the CustomDataDefinition, the event occurs.

```
' Dimension a module variable that hears DiagramType events
Private WithEvents igxDiagramType As DiagramType

Private Sub Main()
    ' Dimension the variables
   Dim igxApp As Application
   Dim igxDiagram As Diagram
   Dim igxShape As Shape
   Dim igxCustomDataDef As CustomDataDefinition
   ' Set the igxApp variable to the current Application object.
   Set igxApp = Application.Application
   ' Get the ActiveDiagram object
   Set igxDiagram = ActiveDiagram
   ' Get the DiagramType object
   Set igxDiagramType = igxDiagram.DiagramType
   ' Add a shape to the diagram
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Add a CustomDataDefinition to the document
   ActiveDocument.CustomDataDefinitions.Add "MyDataField", _
       ixCustomDataFormatTextBase
   ' Pause for user
   MsgBox "New return to iGrafx and try changing a" & _
       "CustomDataDefinition in the shape."
End Sub

' This event fires if a CustomDataDefinition is changed
Private Sub igxDiagramType_CustomDataDefinitionChange()
   MsgBox "A custom data value has changed."
End Sub
```

See Also CustomDataDefinitions object

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

ExternalTemplateDate Property

Syntax *DiagramType*.**ExternalTemplateDate**

Data Type Date string—standard Visual Basic data type (read-only)

Description The ExternalTemplateDate property returns the date and time of the original template file from which the DiagramType originated. This property can be used to check the original template to see if it was modified, or if it is a new version of an older template.

Example The following example creates a diagram from the Cascade template, and then displays the date and time of the template file.

```
' Dimension the variables
Dim igxDiagrams As Diagrams
Dim igxDiagram1 As Diagram
Dim igxDiagramType As DiagramType
' Set the Diagrams object
Set igxDiagrams = ActiveDocument.Diagrams
' Create 2 new diagrams from the Cascade template
MsgBox "Click OK to add a new diagram from the Cascade Template"
Set igxDiagram1 = igxDiagrams.AddFromTemplate("MyCascadeA", _
   "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\Cascade.igt", _
   "Cascade", True)
' Get the DiagramType object
Set igxDiagramType = igxDiagram1.DiagramType
' Display the data of the original template file
MsgBox "This diagram's template was last updated " & _
   igxDiagramType.ExternalTemplateDate
```

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

Initialize Event

Syntax **Private Sub** *DiagramType*_**Initialize**()

Description The Initialize event occurs when a diagram of a particular type is created for the first time. The event provides a way to perform any initial setup of a diagram, interface, or other elements the first time a diagram of a particular type is created. The Initialize event should be used inside a Template file that has been saved to disk.

Example The following example adds a new property to a DiagramType called "TimeInitialized". This user-defined property returns the date and time that the DiagramType was initialized. To try this code, open a Template (.igt) file and put this code into the "ThisDiagramType" code window.

```
Private InitializationTime As String

Private Sub DiagramType_Initialize()
   InitializationTime = Now
End Sub

Public Property Get TimeInitialized()
   TimeInitialized = InitializationTime
End Property
```

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

NotifyWhenTemplateChanges Property

Syntax *DiagramType*.**NotifyWhenTemplateChanges**[= **{True | False}**]

Data Type Boolean (read/write)

Description The NotifyWhenTemplateChanges property specifies whether the user is notified when a template has changed. If a diagram is created from a template, saved to disk, and later the original template is modified, the user is notified of the change the next time the diagram is opened. The user is notified with a message and the option to update the diagram to reflect the changes in the template. If this property is set to False, the user is not notified, and the diagram is not updated. Set the property to True if your program needs to control the synchronization of diagrams with templates.

Example The following example bypasses the Notify message by setting the property to False. It then updates the diagram from the template with no input from the user.' Dimension iGrafx variables

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDiagramType As DiagramType
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the DiagramType object
Set igxDiagramType = igxDiagram.DiagramType
' Switch off the notify message
igxDiagramType.NotifyWhenTemplateChanges = False
' Update the diagram internally if it needs it
If Not igxDiagramType.UpToDateWithTemplate Then
    igxDiagramType.UpdateFromTemplate
End If
```

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

PluralName Property

Syntax *DiagramType*.**PluralName**

Data Type String (read-only)

Description The PluralName property returns the plural form of the DiagramType name.

Example The following example displays the plural form of a DiagramType name.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
' Get the DiagramType object
Set igxDiagramType = ActiveDiagram.DiagramType
' Display the name of the DiagramType
MsgBox "Diagrams of this type are called " & igxDiagramType.PluralName
```

See Also SingularName property

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

ProgID Property

Syntax *DiagramType*.**ProgID**

Data Type String (read-only)

Description The ProgID property returns a string containing the application name, and the type of diagram for the specified DiagramType object (for example "iGrafx.BasicDiagram").

Example The following example displays the ProgID of a DiagramType object.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxDiagram As Diagram
' Get the ActiveDiagram object
Set igxDiagram = ActiveDiagram
' Get the DiagramType object
Set igxDiagramType = igxDiagram.DiagramType
' Display the name of the DiagramType
MsgBox "This DiagramType's ProgID: " & igxDiagramType.ProgID
```

See Also ClassID property

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

SetTemplate Method

Syntax *DiagramType*.**SetTemplate**(*TemplateName* As String, *DiagramInTemplate* As String, [*UpdateNow* As Boolean = False])

Description The SetTemplate method specifies which template file a DiagramType is associated with. One possible use of this method is to try multiple versions of a template with one DiagramType.

The *TemplateName* argument specifies the path and file name of a template file (.igt).

The *DiagramInTemplate* argument specifies the name of a diagram within the template to associate with the DiagramType object.

The *UpdateNow* argument specifies whether the diagram is immediately updated to reflect the new template. If set to True, the DiagramType is immediately updated. If set to False, the diagram is not updated. The UpdateFromTemplate method can then be used later to update the DiagramType object.

Example The following example creates a basic diagram from the Cascade template, and uses the SetTemplate method to associate the DiagramType with the Pyramid template.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDiagramType As DiagramType
' Set the Diagrams object
Set igxDiagram = ActiveDocument.Diagrams _
    .AddFromTemplate(ActiveDocument.Diagrams.NextSuggestedName, _
   "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\Cascade.igt", _
   "Cascade")
MsgBox "Diagram added. Click OK to change the template."
' Set the DiagramType to the Cascade template
Set igxDiagramType = igxDiagram.DiagramType
' Set a new template for the DiagramType
igxDiagramType.SetTemplate _
    "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\Pyramid.igt", _
   "Pyramid"
' Pause for user
MsgBox "The DiagramType has been changed to the Pyramid template."
```

See Also UpdateFromTemplate method

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

ShapeLibrary Property

Syntax *DiagramType*.**ShapeLibrary**

Data Type ShapeLibrary object (read-only, See Object Properties)

Description The ShapeLibrary property returns the ShapeLibrary object for the specified DiagramType object. The ShapeLibrary object gives you access to the ShapeLibraryItems it contains (i.e. shapes).

Example The following example displays the name of the ShapeLibrary associated with the DiagramType object.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDiagramType As DiagramType
' Set the Diagrams object
Set igxDiagram = ActiveDocument.Diagrams _
   .AddFromTemplate(ActiveDocument.Diagrams.NextSuggestedName, _
   "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\Cascade.igt", _
   "Cascade")
' Get the DiagramType object
Set igxDiagramType = igxDiagram.DiagramType
' Display the ShapeLibrary for the DiagramType
MsgBox "This DiagramType uses the " & _
   igxDiagramType.ShapeLibrary.CollectionName & _
   " ShapeLibrary subject."
```

See Also ShapeLibrary object

iGrafx API Object Hierarchy

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

SingularName Property

Syntax *DiagramType*.**SingularName**

Data Type String (read-only)

Description The SingularName property returns the singular form of the DiagramType name.

Example The following example displays the singular form of a DiagramType name.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxDiagram As Diagram
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the DiagramType object
Set igxDiagramType = igxDiagram.DiagramType
' Display the name of the DiagramType
MsgBox "A Diagram of this type is called a " & _
    igxDiagramType.PluralName
```

See Also PluralName property

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

SynchronizedToDate Property

Syntax *DiagramType*.**SynchronizedToDate**

Data Type Date string—standard Visual Basic data type (read-only)

Description The SynchronizedToDate property returns the date and time associated with the template to which a DiagramType is synchronized. That is, the value returned is the date and time the template was created or last modified.

Example The following example displays the date and time of the template to which the DiagramType is synchronized.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDiagramType As DiagramType
' Set the Diagrams object
Set igxDiagram = ActiveDocument.Diagrams _
   .AddFromTemplate(ActiveDocument.Diagrams.NextSuggestedName, _
   "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\Cascade.igt", _
   "Cascade")
' Get the DiagramType object
Set igxDiagramType = igxDiagram.DiagramType
' Display the ShapeLibrary for the DiagramType
MsgBox "This DiagramType is synchronized to a template dated " & _
   igxDiagramType.SynchronizedToDate
```

See Also .ExternalTemplateDate property

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

TemplateName Property

Syntax *DiagramType*.**TemplateName**

Data Type String (read-only)

Description The TemplateName property returns the name of the template with which the DiagramType is associated.

Example The following example displays the name of the template with which the DiagramType is associated.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDiagramType As DiagramType
' Set the Diagrams object
Set igxDiagram = ActiveDocument.Diagrams _
   .AddFromTemplate(ActiveDocument.Diagrams.NextSuggestedName, _
   "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\Cascade.igt", _
   "Cascade")
' Get the DiagramType object
Set igxDiagramType = igxDiagram.DiagramType
' Display the template name for the DiagramType
MsgBox "This DiagramType is associated with this template-- " & _
    igxDiagramType.TemplateName
```

See Also SetTemplate method

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

Terminate Event

Syntax **Private Sub** *DiagramType*_**Terminate**()

Description The Terminate event occurs when all instances of the DiagramType are removed from the document. For instance, if the document has two "Basic Diagram" diagrams, and each one is closed, the Terminate event occurs after the last one closes.

Example The following example adds a new property to a DiagramType called "TimeInitialized". This user-defined property returns the date and time that the DiagramType was initialized. To try this code, open a Template (.igt) file and put this code into the "ThisDiagramType" code window.

```
Private InitializationTime As String

Private Sub DiagramType_Initialize()
   InitializationTime = Now
End Sub

Public Property Get TimeInitialized()
   TimeInitialized = InitializationTime
End Property
```

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

UpdateFromTemplate Method

Syntax *DiagramType*.**UpdateFromTemplate**

Description The UpdateFromTemplate method updates a DiagramType from it's template. If a template is modified, use this method to update the DiagramType to reflect the changes.

Example The following example updates the diagram from the template with no input from the user.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxDiagram As Diagram
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the DiagramType object
Set igxDiagramType = igxDiagram.DiagramType
' Switch off the notify message
igxDiagramType.NotifyWhenTemplateChanges = False
' Update the diagram internally if it needs it
If Not igxDiagramType.UpToDateWithTemplate Then
    igxDiagramType.UpdateFromTemplate
End If
```

See Also SetTemplate method

UpToDateWithTemplate property

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

UpToDateWithTemplate Property

Syntax *DiagramType*.**UpToDateWithTemplate**[= **{True | False}**]

Data Type Boolean (read-only)

Description The UpToDateWithTemplate property spedifies whether a DiagramType is up-to-date with it's template. This property, combined with the UpdateFromTemplate method, provide an automated way to update DiagramType objects without input from a user.

Example The following example updates the diagram from the template with no input from the user.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxDiagram As Diagram
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the DiagramType object
Set igxDiagramType = igxDiagram.DiagramType
' Switch off the notify message
igxDiagramType.NotifyWhenTemplateChanges = False
' Update the diagram internally if it needs it
If Not igxDiagramType.UpToDateWithTemplate Then
    igxDiagramType.UpdateFromTemplate
End If
```

See Also UpdateFromTemplate method

{button DiagramType object,JI(`igrafxrf.HLP',`DiagramType_Object')}

DiagramTypes Object

The DiagramTypes object is a collection of individual DiagramType objects. A DiagramTypes collection is associated with and accessible from the Application object and the Document object. Its purpose is to store and provide access to the individual DiagramType objects that are available for use.

The DiagramTypes object provides the following functionality:

- The ability to access any DiagramType objects in the collection.
- The ability to determine how many DiagramType objects are in the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the DiagramTypes object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
Parent		

Item Method

Syntax *DiagramTypes*.**Item**(Index) As DiagramType

Description The Item method returns a DiagramType object from the DiagramTypes collection. The *Index* argument specifies which object to return.

There is a DiagramTypes collection at the application level, and at the document level.

At the application level, the collection contains exactly two items. The *Index* argument can be 1 or 2. Item(1) always returns the Process diagram type object, and Item(2) always returns the Basic Diagram type object.

At the application level the collection is fixed, but at the document level it is not. At the document level, the contents of the DiagramTypes collection depends on whether one or both diagram types are being used in the document, and the order in which they were created. The collection can contain two items at most, and only if both types of diagrams are being used in the document.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example displays the plural name of all the DiagramType objects in the DiagramTypes collection, accessed from the Application object.

```
' Dimension the variables
Dim sString As String
' Collect the names of the DiagramType objects into a string
For Index = 1 To Application.DiagramTypes.Count
    sString = sString & Application.DiagramTypes.Item(Index) _
        .PluralName & Chr(13)
Next Index
' Display the result
MsgBox "The Application contains these DiagramType objects:" & _
    Chr(13) & Chr(13) & sString
```

{button DiagramTypes object,JI(`igrafxrf.HLP',`DiagramTypes_Object')}

DiagramObject Object

The DiagramObject object is an “Extender” object for several other objects in the iGrafx Professional object model. A DiagramObject object always has a “type” which identifies the “Primary” object. Together, the exytender object and the primary object make up what is known as a “composite control” in VBA. The DiagramObject object has the following Primary objects associated wirth it:

- ConnectorLine object
- Department object
- Group object
- Legend object
- OLEObject object
- Shape object
- TextGraphicObject object

The DiagramObject object contains properties, methods, and events that are common to all of the primary objects. The primary objects have properties, methods, and events that are specific to that type of object.

When working at the DiagramObject object level, sometimes it is important to know the type of a particular DiagramObject, and sometimes it is not. To access the Primary object level, you do have to know the type; otherwise, your code can generate errors.

Properties, Methods, and Events

All of the properties, methods, and events for the DiagramObject object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Angle	CreateVbaControl	AfterChangeLayer
Application	DeleteDiagramObject	AfterEditCustomData
AsType	DeleteVbaControl	AfterFontChange
AttachedObjects	FireUserEvent	AfterGroup
Bottom	MakeObjectRange	AfterMove
CenterX	Move	AfterRotate
CenterY	Redraw	AfterSave
ConnectorLine	Resize	AfterSize
CustomDataValues	UpdateFields	AfterStyleChange
Department		AfterTextChange
Diagram		AfterUngroup
Fields		BeforeChangeLayer
Group		BeforeClick
Height		BeforeDelete
ID		BeforeDoubleClick
IsGrouped		BeforeEditCustomData
IsVBAControl		BeforeFontChange
Layer		BeforeGroup
Left		BeforeMove
Legend		BeforeRightClick
Name		BeforeRotate
Object		BeforeSave
ObjectName		BeforeSelect

OleObject
Parent
PermanentDiagramObject
PropertyLists
Right
Selected
Shape
TextGraphicObject
Top
Type
Width

BeforeSize
BeforeStyleChange
BeforeTextChange
BeforeUngroup
Close
ContextMenu
DeleteObject
Deselect
GetInterface
Load
Modify
New
PropertyChange
Select
UserEvent

Related Topics

ConnectorLine object
Department object
Group object
Legend object
OLEObject object
Shape object
TextGraphicObject object

AfterChangeLayer Event

Syntax **Private Sub** *DiagramObject*_**AfterChangeLayer**(*NewLayer* As Layer)

Description The AfterChangeLayer event occurs after the specified DiagramObject object has been moved to a different layer of a diagram, either interactively by a user or programmatically. The *NewLayer* parameter contains the Layer object to which the DiagramObject was moved.

Example The following example moves a shape to a new layer, which fires the AfterChangeLayer event. The event subroutine then displays a message containing the name of the layer to which the shape was moved.

```
' Dimension a module variable that hears events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
   Dim igxShape As Shape
   Dim igxLayer1 As Layer
   Dim igxLayer2 As Layer
   ' Add a shape to the active diagram
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Get the existing layer object
   Set igxLayer1 = ActiveDiagram.ActiveLayer
   ' Add a new layer
   Set igxLayer2 = ActiveDiagram.Layers.Add("Layer B")
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape.DiagramObject
   ' Move the shapes in layer 1 to layer 2
   MsgBox "Click OK to move the shape up one layer."
   igxLayer1.ObjectRange.MoveToLayer 2
End Sub

Private Sub igxDiagramObject_AfterChangeLayer(ByVal NewLayer As Layer)
   ' Display a message if the diagram object moves to a new layer
   MsgBox igxDiagramObject.Name & " has moved to layer: " _
       & NewLayer.Name
End Sub
```

See Also BeforeChangeLayer event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AfterEditCustomData Event

Syntax **Private Sub** *DiagramObject*_**AfterEditCustomData**(*NewFieldValue* As CustomDataValue)

Description The AfterEditCustomData event occurs after the specified DiagramObject object's CustomDataValue object has been changed or modified. This event fires every time the value is changed, including the first time a value is specified. This is because, technically, any change from the initialized value, whether it is an empty string, or zero, or some other value, is considered a change by the event.

The *NewFieldValue* parameter contains the new value of the CustomDataValue object that was changed.

Example The following example creates a shape and changes a CustomDataValue object in the shape. This triggers the event, which displays the new value.

```
' Dimension a module variable that hears events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
   Dim igxShape As Shape
   ' Set a shape object
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape.DiagramObject
    ' Add a CustomDataField to the document
    ActiveDocument.CustomDataDefinitions.Add MyField, _
        ixCustomDataFormatTextBase
   ' Set the CustomData Text Field of the shape
   igxDiagramObject.CustomDataValues.Item _
        (1, ixCustomDataText).Value = "My data"
   ' Change the CustomData Text Field of the shape
   MsgBox "Click OK to change the CustomDataValue in the shape."
   igxDiagramObject.CustomDataValues.Item _
        (1, ixCustomDataText).Value = "My other data"
End Sub

Private Sub igxDiagramObject_AfterEditCustomData(ByVal NewFieldValue As
CustomDataValue)
    ' Display the result
   MsgBox "The CustomDataValue was changed to: " & NewFieldValue
End Sub
```

See Also BeforeEditCustomData event

CustomDataValue object

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AfterFontChange Event

Syntax **Private Sub** *DiagramObject*_**AfterFontChange**

Description The AfterFontChange event occurs after any change occurs to the Font object associated with the specified DiagramObject object. Note that the DiagramObject object itself, does not have a Font property. The Font property is associated with particular “Primary” objects, such as Shape, Department, and TextGraphicObject.

Example The following example creates a TextGraphicObject object in the active diagram. It then changes the font on the object, which fires the DiagramObject object’s AfterFontChange event.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
   Dim igxText As TextGraphicObject
   Dim igxFont As Font
   ' Set a text object
   Set igxText = ActiveDiagram.DiagramObjects.AddTextObject _
       (1440, 1440, , , "Text in a TextGraphicObject.")
   ' Set the diagram object variable
   Set igxDiagramObject = igxText.DiagramObject
   MsgBox "Click OK to change the font."
   ' Change the font
   igxDiagramObject.TextGraphicObject.TextRange().Font.Name = _
       Application.FontNames.Item(2)
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_AfterFontChange()
   MsgBox "The object’s Font was changed."
End Sub
```

See Also BeforeFontChange event

Font object

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AfterGroup Event

Syntax **Private Sub** *DiagramObject*_**AfterGroup**(*Group* As Group)

Description The AfterGroup event occurs after the specified DiagramObject object is added to a group. The *Group* parameter contains the Group object to which the DiagramObject was added.

Example The following example creates two shapes in the active diagram, and then adds them to a Group object, which fires the AfterGroup event.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxObjectRange As ObjectRange
   Dim igxFont As Font
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
   Set igxObjectRange = ActiveDiagram.MakeObjectRange
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to create the group."
   ' Add the shapes to the ObjectRange
   igxObjectRange.Add igxShape1.DiagramObject
   igxObjectRange.Add igxShape2.DiagramObject
   ' Make a Group from the ObjectRange
   igxObjectRange.Group
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_AfterGroup(ByVal Group As IXGroup)
   MsgBox "The new Group contains " & Group.ObjectRange.Count _
       & " objects."
End Sub
```

See Also BeforeGroup event

Group object

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AfterMove Event

Syntax **Private Sub** *DiagramObject*_**AfterMove**(ByVal *Left* As Double, ByVal *Top* As Double)

Description The AfterMove event occurs after the specified DiagramObject object has been moved within a diagram. The *Left* and *Top* parameters specify the new position of the DiagramObject.

Example The following example creates two shapes in the active diagram, and then moves the shapes. Moving the shapes triggers the AfterMove event, once for each shape.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxFont As Font
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to move the shapes."
   ' Move the shapes by altering the CenterY values
   igxShape1.DiagramObject.CenterY = 1440 * 3
   igxShape2.DiagramObject.CenterY = 1440 * 3
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_AfterMove(ByVal Left As Double, ByVal Top As
Double)
    MsgBox "Shape2's left side is at : " & Left & Chr(13) & _
        "and the top is at: " & Top
End Sub
```

See Also BeforeMove event

{button DiagramObject object,JI(`igrafxrf.HLP','DiagramObject_Object')}

AfterRotate Event

Syntax **Private Sub** *DiagramObject*_**AfterRotate**(ByVal *Angle* As Double)

Description The AfterRotate event occurs after the specified DiagramObject object is rotated. The *Angle* parameter specifies the angle of rotation of the DiagramObject after it has been rotated.

Example The following example creates two shapes in the active diagram, and rotates each shape 30 degrees. Rotating the shapes triggers the AfterRotate event, once for each shape.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxFont As Font
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440, 1440 * 3)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to rotate the shapes."
   ' Rotate each shape 30 degrees
   igxShape1.DiagramObject.Angle = 30
   igxShape2.DiagramObject.Angle = 30
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_AfterRotate(ByVal Angle As Double)
   MsgBox "The shape has been rotated " & (Angle) & " degrees."
End Sub
```

See Also BeforeRotate event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AfterSave Event

Syntax **Private Sub** *DiagramObject*_**AfterSave**()

Description The AfterSave event occurs after the document that contains the specified DiagramObject object has been saved.

Example The following example creates two shapes in the active diagram of a document. The document is then saved, which triggers the AfterSave event. The event is fired only for Shape2. Be sure to use a valid file system path for the SaveDocumentAs method in the code.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to save the document."
   ActiveDocument.SaveDocumentAs "E:\My Documents\test.igx"
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_AfterSave()
   MsgBox "The document containing the DiagramObject was saved."
End Sub
```

See Also BeforeSave event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AfterSize Event

Syntax **Private Sub** *DiagramObject*_**AfterSize**(ByVal *Width* As Double, ByVal *Height* As Double)

Description The AfterSize event occurs after the specified DiagramObject object has had its size changed. The *Width* parameter specifies the new width of the DiagramObject, and the *Height* parameter specifies the new height.

Example The following example creates a new shape in the active diagram, and then resizes the shape. Resizing the shape triggers the AfterSize event, which displays a message indicating the new size of the shape.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxFont As Font
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
   ' Set the DiagramObject variable
   Set igxDiagramObject = igxShape1.DiagramObject
   MsgBox "Click OK to resize the the shape."
   ' Resize the object
   igxDiagramObject.Width = 2500
   igxDiagramObject.Height = 3000
   ' Pause for the user
   MsgBox "The event fired twice, once for each line of " _
       & "code performing a resize."
End Sub

Private Sub igxDiagramObject_AfterSize(ByVal Width As Double, ByVal Height As
Double)
   MsgBox "The shape is now " & Width & " X " & Height
End Sub
```

See Also BeforeSize event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AfterStyleChange Event

Syntax **Private Sub** *DiagramObject*_**AfterStyleChange**()

Description The AfterStyleChange event occurs after the style of the specified DiagramObject has been changed. A style change is any change to an appearance attribute of a DiagramObject, such as fill color, line style, arrow size, etc.

Example The following example treats the diagram as a class, and adds a property “Get” to the diagram called ObjectsLastAltered. This property returns the date and time when a DiagramObject was last altered in the diagram. The "privateLastAltered" variable stores the date and time. The AfterStyleChange event assigns the date and time to the variable. The ObjectsLastAltered property returns the value of the variable. Other programs can then reference this diagram, and retreive it's ObjectsLastAltered property. Included at the bottom is a routine that sets up two shapes and a connector line. It changes the line style of the connector line to fire the event.

```
' Module variable for storing date and time
Private privateLastAltered As Date

' New class property returns last time a
' DiagramObject was altered on this diagram
Public Property Get ObjectsLastAltered() As Date
   ObjectsLastAltered = privateLastAltered
End Property

' This event stores the date and time whenever a
' DiagramObject style is changed.
Private Sub AnyObject_AfterStyleChange()
   privateLastAltered = Now
   MsgBox "The style of a DiagramObject has changed."
End Sub

' Test routine makes a style change
Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxConnector As ConnectorLine
   ' Add two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 5, 1440)
   ' Add a connector line
   Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
   MsgBox "Click OK to change the style of the connector line."
   ' Change the line style of the connector line
   igxConnector.LineStyle = ixLineDashed
   MsgBox "Click OK to continue."
End Sub
```

See Also BeforeStyleChange event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AfterTextChange Event

Syntax **Private Sub** *DiagramObject*_**AfterTextChange**(*PreviousText* As String)

Description The AfterTextChange event occurs after any text associated with the specified DiagramObject object is changed. The *PreviousText* parameter contains the text of the DiagramObject before it was changed.

Example The following example creates a shape in the active diagram, and then changes the text of the shape. This fires the AfterTextChange event, which displays the old text and the new text.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   ' Create a shape
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Switch to the DiagramObject level
   Set igxDiagramObject = igxShape1.DiagramObject
   ' Set the text of the shape
   igxDiagramObject.Shape.Text = "Activity 1"
   MsgBox "Click OK to change the text."
   ' Change the text
   igxDiagramObject.Shape.Text = "Activity A"
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_AfterTextChange(ByVal PreviousText As String)
    MsgBox "The text on the object used to read:" & Chr(13) & _
       PreviousText & Chr(13) & Chr(13) & _
       "Now the text reads:" & Chr(13) & _
       igxDiagramObject.Shape.Text
End Sub
```

See Also BeforeTextChange event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AfterUngroup Event

Syntax **Private Sub** *DiagramObject*_**AfterUngroup**()

Description The AfterUngroup event occurs after the specified DiagramObject has been ungrouped from its association in a group.

Example The following example creates two objects and adds them to a Group. It then ungroups the objects, which fires the event.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxObjectRange As ObjectRange
   Dim igxGroup As Group
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440, 1440 * 3)
   Set igxObjectRange = ActiveDiagram.MakeObjectRange
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to create the group."
   ' Add the shapes to the ObjectRange
   igxObjectRange.Add igxShape1.DiagramObject
   igxObjectRange.Add igxShape2.DiagramObject
   ' Make a Group from the ObjectRange
   Set igxGroup = igxObjectRange.Group
   MsgBox "Click OK to ungroup the objects."
   ' Ungroup the objects
   igxGroup.Ungroup
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_AfterUngroup()
   MsgBox "The object has been removed from a group."
End Sub
```

See Also BeforeUngroup event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Angle Property

Syntax *DiagramObject*.**Angle**

Data Type Double (read/write)

Description The Angle property rotates the specified DiagramObject object. The rotation occurs around the object's center (the location specified by the CenterX and CenterY properties). The unit value for this property is degrees of rotation; that is, 0 to 359 (360 is the same as 0). Values above 359 are legal, as are negative values. A positive value rotates the object clockwise, and a negative value rotates the object counterclockwise.

Example The following example creates two shapes on the active diagram and draws a connector line between them. Then the first shape is rotated, using the Angle property within a For loop, through 360 degrees in 15 degree increments. A message box is displayed at each increment.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxDiagObj As DiagramObject
Dim iCount As Integer
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Get the DiagramObject level for Shape 1
Set igxDiagObj = igxShape1.DiagramObject
' Observe shape location and connector line attachment as
' Shape 1 is rotated through 360 degrees
MsgBox "Check location of Shape 1"
For iCount = 1 To 24
    igxDiagObj.Angle = iCount * 15
    MsgBox ("Check location. Shape 1 rotated " & iCount * 15 _
        & " degrees")
Next iCount
```

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AsType Property

Syntax *DiagramObject*.**AsType**(*TypeName* As String) As Object

Data Type An Object of the type identified by the *TypeName* argument (read/write)

Description The AsType property allows you to add your own properties and methods to a document object, extending the object model. The properties and methods can be organized into one or more document types, using unique type names.

The *TypeName* argument is a string that names the custom type. It can be any string you choose, but it must be unique within the environment. In an integrated environment, other programmers may be accessing the document, and using it's AsType property. To prevent conflicting type names, it is suggested that you use your company or department name, followed by a descriptive type name (for example, "MyCompanyFactory")

Use the following basic steps to implement a custom property or method for the Document object.

1. Use Document.AsType ("my type name").MyMethod in your code.
2. Create a new Class, and design properties and methods in the class.
3. Set up the GetInterface event to check the TypeName string passed to it. If it matches your type name, set the Interface parameter equal to your new class.

When you use Document.AsType(*TypeName*) in your code, you gain access to the properties and methods that you have defined in the new Class. The Document.AsType property automatically fires an event called GetInterface. The GetInterface event can have one or more AsType's defined, each one distinguished by a unique type name. Based on the type name, the GetInterface event redirects execution to your new Class by setting the Interface parameter. If the Interface parameter is set to your new Class, the Class properties and methods become exposed to the Document object.

Example Using the AsType property, the GetInterface event, and VBA's support for Classes, you can extend key iGrafx objects. The following example creates a shape AsType Airplane, and retreives it's AircraftType property.

The first step to doing this is creating a VBA class. The following code defines a simple class which has two properties—AirCraftType, and Departure.

Insert a new class called Class1, and copy this block of code into it.

```
' Class
Public Property Get AirCraftType() As String
   AirCraftType = "Boeing 747"
End Property

Public Property Get Departure() As String
   Departure = "Monday, 8:05 AM"
End Property
```

The following two blocks of code go in the project code window above the new Class1.

```
' Run this to test the event
Private Sub Main()
   Dim igxShape As Shape
   Dim igxDiagramObject As DiagramObject
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxDiagramObject = igxShape.DiagramObject
   MsgBox "The aircraft is a " & _
```

```
        igxDiagramObject.AsType("Airplane").AirCraftType
End Sub

' The GetInterface event is fired whenever the AsType method is used.
' Based on the TypeName, redirect the interface to your custom class
Private Sub AnyObject_GetInterface(ByVal TypeName As String, Interface As
Object)
    ' If the broadcast type name is "Airplane", then set the interface
   If TypeName = "Airplane" Then
        ' TypeName gets broadcast everywhere, so we need to check if
        ' something else grabbed and set the Interface first
        If Interface Is Nothing Then
            Set Interface = New Class1
        Else
            MsgBox "ERROR: Someone else is using Airplane AsType"
        End If
   End If
End Sub
```

See Also GetInterface event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

AttachedObjects Property

Syntax *DiagramObject*.**AttachedObjects**

Data Type ObjectRange object (read-only, See Object Properties)

Description The AttachedObjects property returns an ObjectRange object that contains all the TextGraphicObject objects that are attached to the DiagramObject.

Example The following example creates a shape and attaches a TextGraphicObject to it. It then changes the color of the text object, accessing it through the AttachedObjects property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxText1 As TextGraphicObject
' Create a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Create a TextGraphicObject
Set igxText1 = ActiveDiagram.DiagramObjects.AddTextObject _
    (1440 * 2, 1440 * 2)
' Attach the text object to the shape
igxText1.AttachTo igxShape1.DiagramObject
' Add text to the text object
igxText1 = "This text is attached to the shape."
MsgBox "Click OK to change the color of TextGraphicObjects " _
    & "attached to the shape."
' Change the color of text objects attached to the shape
igxShape1.DiagramObject.AttachedObjects.FillFormat.FillColor = vbGreen
MsgBox "Click OK to continue."
```

See Also ObjectRange object

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeChangeLayer Event

Syntax **Private Sub** *DiagramObject*_**BeforeChangeLayer**(*NewLayer* As Layer, *Cancel* As Boolean)

Description The BeforeChangeLayer event occurs before the specified DiagramObject object has been moved to a different layer of a diagram, either interactively by a user or programmatically. The *NewLayer* parameter contains the Layer object to which the DiagramObject is about to move. The *Cancel* parameter allows you to cancel the move. If set to True, the layer change is canceled. If set to False (default), the layer change takes affect.

Example The following example moves a shape to a new layer, which fires the AfterChangeLayer event. The event subroutine then displays a message containing the name of the layer to which the shape was moved.

```
' Dimension a module variable that hears events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
   Dim igxShape As Shape
   Dim igxLayer1 As Layer
   Dim igxLayer2 As Layer
   ' Add a shape to the active diagram
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Get the existing layer object
   Set igxLayer1 = ActiveDiagram.ActiveLayer
   ' Add a new layer
   Set igxLayer2 = ActiveDiagram.Layers.Add("Layer B")
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape.DiagramObject
   ' Move the shapes in layer 1 to layer 2
   MsgBox "Click OK to move the shape up one layer."
   igxLayer1.ObjectRange.MoveToLayer 2
End Sub

Private Sub igxDiagramObject_BeforeChangeLayer(ByVal NewLayer As Layer, Cancel
As Boolean)
   ' Cancel the layer change if the user responds with No
   If MsgBox("The object is about to be moved to layer: " _
   & NewLayer & Chr(13) & "Allow the move?", vbYesNo) _
   = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also AfterChangeLayer event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeClick Event

Syntax **Private Sub** *DiagramObject*_**BeforeClick**(ByVal *X* As Double, ByVal *Y* As Double, *Cancel* As Boolean)

Description The BeforeClick event occurs when a DiagramObject is being clicked on with the mouse. The event allows you to intercept the mouse click and perform some action or actions before anything else happens.

The *X* and *Y* parameters provide the position of the mouse cursor within the DiagramObject at the time of the click. The values returned are determined by the local coordinate space of the DiagramObject, usually a decimal value between 0.0 and 1.0. The *Cancel* parameter allows you to cancel the mouse click. If set to True, the click is canceled, as if the click never occurred. If set to False (default), the event code is run, followed by whatever else has been programmed to happen to the DiagramObject in response to a single click.

Example The following example sets up the BeforeClick event with a shape, and gives the user the option to cancel the click.

```
' Dimension a variable that listens to DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape As Shape
   ' Add a shape object
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape.DiagramObject
   MsgBox "Return to the diagram and click on the shape."
End Sub

Private Sub igxDiagramObject_BeforeClick(ByVal X As Double, ByVal Y As Double,
Cancel As Boolean)
   ' Cancel the click if the user responds with No
   If MsgBox("The object is being clicked at " & X & "," & Y _
   & Chr(13) & "Allow the click?", vbYesNo) _
   = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also BeforeDoubleClick event

BeforeRightClick event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeDelete Event

Syntax **Private Sub** *DiagramObject*_**BeforeDelete**(*Cancel* As Boolean)

Description The BeforeDelete event occurs before the specified DiagramObject is deleted. The *Cancel* parameter allows you to cancel the deletion, and can be set by the programmer before the event ends. If set to True, the deletion is canceled. If set to False (default), the DiagramObject is deleted when the event ends.

Example The following example deletes a shape, which triggers the BeforeDelete event. The user is given the option to cancel the deletion.

```
' Dimension a variable that listens to DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape As Shape
   ' Add a shape object
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape.DiagramObject
   MsgBox "Click OK to delete the shape."
   igxShape.DiagramObject.DeleteDiagramObject
End Sub

Private Sub igxDiagramObject_BeforeDelete(Cancel As Boolean)
   ' Cancel if the user responds with No
   If MsgBox("The object is about to be deleted." _
   & Chr(13) & "Proceed with the delete?", vbYesNo) _
   = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also DeleteObject event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeDoubleClick Event

Syntax **Private Sub** *DiagramObject*_**BeforeDoubleClick**(ByVal *X* As Double, ByVal *Y* As Double, *Cancel* As Boolean)

Description The BeforeDoubleClick event occurs as the specified DiagramObject is being double clicked with the mouse. The event allows you to intercept the mouse double click and perform some action or actions before anything else happens.

The *X* and *Y* parameters provide the position of the mouse cursor within the DiagramObject at the time of the double click. The values returned are determined by the local coordinate space of the DiagramObject, usually a decimal value between 0.0 and 1.0. The *Cancel* parameter allows you to cancel the double click. If set to True, the double click is canceled, as if it never occurred. If set to False (default), the event code is run, followed by whatever else has been programmed to happen to the DiagramObject in response to a double click.

When the user double clicks the mouse, both the "Click" and "Double Click" events are fired, because a double click is treated as a single click, followed by a rapid second click.

Example The following example sets up the BeforeDoubleClick event with a shape, and gives the user the option to cancel the double click.

```
' Dimension a variable that listens to DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape As Shape
   ' Add a shape object
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape.DiagramObject
   MsgBox "Return to the diagram and click on the shape."
End Sub

Private Sub igxDiagramObject_BeforeClick(ByVal X As Double, ByVal Y As Double,
Cancel As Boolean)
   ' Cancel the click if the user responds with No
   If MsgBox("The object is being clicked at " & X & "," & Y _
   & Chr(13) & "Allow the click?", vbYesNo) _
   = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also BeforeClick event

BeforeRightClick event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeEditCustomData Event

Syntax **Private Sub** *DiagramObject*_**BeforeEditCustomData**(*FieldValue* As CustomDataValue, *Cancel* As Boolean)

Description The BeforeEditCustomData event occurs as the specified DiagramObject object's CustomDataValue object is changed or modified. This event fires every time the value is changed, including the first time a value is specified. This is because, technically, any change from the initialized value, whether it is an empty string, or zero, or some other value, is considered a change by the event.

The *FieldValue* parameter contains the value contained in the field before the change was specified; that is, it contains the old value, not the new one. The programmer can set the *Cancel* parameter to True to cancel the change.

Example The following example changes a CustomDataValue of a DiagramObject, which fires the event. The user is given the option to cancel the change.

```
' Dimension a module variable that hears events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape As Shape
   ' Set a shape object
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable for the shape
   Set igxDiagramObject = igxShape.DiagramObject
   ActiveDocument.CustomDataDefinitions.Add _
       MyField, ixCustomDataFormatTextBase
   ' Set the CustomData Text Field of the shape
   igxDiagramObject.CustomDataValues.Item _
       (1, ixCustomDataText).Value = "My data"
   MsgBox "MyField contains the value, " _
       & igxDiagramObject.CustomDataValues.Item _
       (1, ixCustomDataText).Value
   ' Change the CustomData Text Field of the shape
   MsgBox "Click OK to change the CustomDataValue in the shape."
   igxDiagramObject.CustomDataValues.Item _
       (1, ixCustomDataText).Value = "My other data"
    MsgBox "MyField contains the value, " _
       & igxDiagramObject.CustomDataValues.Item _
       (1, ixCustomDataText).Value
End Sub

Private Sub igxDiagramObject_BeforeEditCustomData(ByVal FieldValue As
CustomDataValue, Cancel As Boolean)
   If (FieldValue = "") Then
        If MsgBox("The CustomDataValue is about to be changed" _
        & Chr(13) & "from its initial value." & Chr(13) _
        & "Proceed with the change?", vbYesNo) = vbNo Then
            MsgBox "Data was not changed"
            Cancel = True
        Else
            MsgBox "Data was changed"
        End If
```

```
    Else
         If MsgBox("The CustomDataValue is about to be changed " _
         & "from its previous value of: " & FieldValue _
         & Chr(13) & "Proceed with the change?", vbYesNo) = vbNo Then
             MsgBox "Data was not changed"
             Cancel = True
         Else
             MsgBox "Data was changed"
         End If
    End If
End Sub
```

See Also AfterEditCustomData event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeFontChange Event

Syntax **Private Sub** *DiagramObject*_**BeforeFontChange**(*Cancel* As Boolean)

Description The BeforeFontChange event occurs when any change occurs to the Font object associated with the specified DiagramObject object. Note that the DiagramObject object itself, does not have a Font property. The Font property is associated with particular “Primary” objects, such as Shape, Department, and TextGraphicObject. The programmer can set the *Cancel* parameter to True to cancel the change.

Example The following example changes the font in a DiagramObject, which fires the event. The user is given the option to cancel the change.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxText As TextGraphicObject
   Dim igxFont As Font
   ' Set a text object
   Set igxText = ActiveDiagram.DiagramObjects.AddTextObject _
       (1440, 1440, , , "Text in a TextGraphicObject.")
   ' Set the diagram object variable
   Set igxDiagramObject = igxText.DiagramObject
   MsgBox "Click OK to change the font."
   ' Change the font
   igxDiagramObject.TextGraphicObject.TextRange().Font.Name = _
       Application.FontNames.Item(2)
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_BeforeFontChange(Cancel As Boolean)
   If MsgBox("The font is about to be changed." _
   & FieldValue & Chr(13) & "Proceed with the change?", _
   vbYesNo) = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also AfterFontChange event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeGroup Event

Syntax **Private Sub** *DiagramObject*_**BeforeGroup**(*Cancel* As Boolean)

Description The BeforeGroup event occurs as the specified DiagramObject object is added to a group. The programmer can set the *Cancel* parameter to True to cancel the addition of the DiagramObject to a group.

Example The following example attempts to add a shape to a Group, which fires the event. The user is then given the option to cancel the grouping.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxObjectRange As ObjectRange
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
   Set igxObjectRange = ActiveDiagram.MakeObjectRange
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to create the group."
   ' Add the shapes to the ObjectRange
   igxObjectRange.Add igxShape1.DiagramObject
   igxObjectRange.Add igxShape2.DiagramObject
   ' Make a Group from the ObjectRange
   igxObjectRange.Group
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_BeforeGroup(Cancel As Boolean)
   If MsgBox("The object is being added to a Group." _
   & FieldValue & Chr(13) & "Proceed with the Group?", _
   vbYesNo) = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also AfterGroup event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeMove Event

Syntax **Private Sub** *DiagramObject*_**BeforeMove**(*Cancel* As Boolean)

Description The BeforeMove event occurs as the specified DiagramObject object is moved within a diagram. The programmer can cancel the change to the DiagramObject by setting the *Cancel* parameter to True.

Example The following example attempts to move a shape, which fires the event. The user is then given the option to cancel the move.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to move the shapes."
   ' Move the shapes by altering the CenterY values
   igxShape1.DiagramObject.CenterY = 4000
   igxShape2.DiagramObject.CenterY = 4000
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_BeforeMove(Cancel As Boolean)
   If MsgBox("The object is about to be moved." _
   & FieldValue & Chr(13) & "Proceed with the move?", _
   vbYesNo) = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also AfterMove event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeRightClick Event

Syntax **Private Sub** *DiagramObject*_**BeforeRightClick**(ByVal *X* As Double, ByVal *Y* As Double, *Cancel* As Boolean)

Description The BeforeRightClick event occurs as the specified DiagramObject is being clicked with the right mouse button. The event allows you to intercept the right click and perform some action or actions before anything else happens.

The *X* and *Y* parameters provide the position of the mouse cursor within the shape at the time of the click. The values returned by X and Y are determined by the local coordinate space of the shape, usually a decimal value between 0.0 and 1.0. The programmer can cancel the right click by setting the *Cancel* parameter to True.

Example The following example sets up the BeforeRightClick event with a shape, and gives the user the option to cancel the right-click.

```
' Dimension a variable that listens to DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape As Shape
   ' Add a shape object
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape.DiagramObject
   MsgBox "Return to the diagram and click on the shape."
End Sub

Private Sub igxDiagramObject_BeforeRightClick(ByVal X As Double, ByVal Y As
Double, Cancel As Boolean)
   ' Cancel the click if the user responds with No
   If MsgBox("The object is being clicked at " & X & "," & Y _
   & Chr(13) & "Allow the click?", vbYesNo) _
   = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also BeforeClick event

BeforeDoubleClick event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeRotate Event

Syntax **Private Sub** *DiagramObject*_**BeforeRotate**(*Cancel* As Boolean)

Description The BeforeRotate event occurs before the specified DiagramObject object is rotated. The programmer can cancel the rotation by setting the *Cancel* parameter to True.

Example The following example attempts to rotate a shape, which fires the event. The user is given the option to cancel the rotation.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to rotate the shapes."
   ' Rotate each shape 30 degrees
   igxShape1.DiagramObject.Angle = 30
   igxShape2.DiagramObject.Angle = 30
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_BeforeRotate(Cancel As Boolean)
   ' If the user answers "No", cancel the change
   If MsgBox("The object is about to be rotated." _
   & FieldValue & Chr(13) & "Proceed with rotation?", _
   vbYesNo) = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also AfterRotate event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeSave Event

Syntax **Private Sub** *DiagramObject*_**BeforeSave**()

Description The BeforeSave event occurs before the document that contains the specified DiagramObject object is saved. The event allows you to intercept the Save command before it happens and perform some action or actions.

Example The following example saves the document. This BeforeSave event is fired, which stores the date and time of the save. Be sure to use a valid file system path for the SaveDocumentAs method in the code.

```
' Dimension a module variable that hears DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject
' Dimension a module variable to store date and time
Private WhenShapeSaved As String

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to save the document."
 ActiveDocument.SaveDocumentAs "E:\My Documents\test.igx"
   ' Pause for the user
   MsgBox "The DiagramObject was last saved to disk " & WhenShapeSaved
End Sub

Private Sub igxDiagramObject_BeforeSave()
   ' Store the date and time the shape was saved to disk
   WhenShapeSaved = Now
End Sub
```

See Also AfterSave event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeSelect Event

Syntax **Private Sub** *DiagramObject*_**BeforeSelect**(*Cancel* As Boolean)

Description The BeforeSelect event occurs before the specified DiagramObject object is selected. The programmer can cancel the selection by setting the *Cancel* parameter to True.

Example The following example attempts to select a shape, which fires the event. The user is given the option to cancel the selection.

```
' Dimension a module variable that hears DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   ' Create a shape
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape1.DiagramObject
   MsgBox "Click OK to select the shape."
   ' Select the shape
   ActiveDiagram.Selection.Add igxDiagramObject
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_BeforeSelect(Cancel As Boolean)
   ' If the user answers "No", cancel the change
   If MsgBox("The object is about to be selected." _
   & FieldValue & Chr(13) & "Proceed with the selection?", _
   vbYesNo) = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also Select event

Selected property

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeSize Event

Syntax **Private Sub** *DiagramObject*_**BeforeSize**(*Cancel* As Boolean)

Description The BeforeSize event occurs before the specified DiagramObject object is sized. The programmer can cancel any change to a DiagramObject object's size by setting the *Cancel* parameter to True.

Example The following example attempts to resize a shape, which fires the event. The user is given the option to cancel the resize.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape1.DiagramObject
   MsgBox "Click OK to resize the the shape."
   ' Resize the object
   igxDiagramObject.Width = 2500
   igxDiagramObject.Height = 3000
   ' Pause for the user
   MsgBox "The event fired twice, once for each time the shape " _
        & "was resized."
End Sub

Private Sub igxDiagramObject_BeforeSize(Cancel As Boolean)
   ' If the user answers "No", cancel the change
   If MsgBox("The object is about to be resized." _
   & FieldValue & Chr(13) & "Proceed with the resize?", _
   vbYesNo) = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also AfterSize event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeStyleChange Event

Syntax **Private Sub** *DiagramObject*_**BeforeStyleChange**(*Cancel* As Boolean)

Description The BeforeStyleChange event occurs before any style change is processed for the specified DiagramObject object. A style change is any change to the appearance of an attribute of a DiagramObject, such as fill color, line style, arrow size, etc. The style change can be canceled by the programmer by setting the *Cancel* parameter to True.

Example The following example uses the AnyObject object to monitor the BeforeStyleChange event. If any DiagramObject has a style change, the event displays a message that asks the user to confirm the change. If the user answers "No", the change is canceled. The Main() subroutine sets up two shapes and a connector line. It then atempts to change the line style of the connector line, which triggers the event.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxConnector As ConnectorLine
   ' Add two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 5, 1440)
   ' Add a connector line
   Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
   MsgBox "Click OK to change the style of the connector line."
   ' Change the line style of the connector line
   igxConnector.LineStyle = ixLineDashed
   MsgBox "Click OK to continue."
End Sub

Private Sub AnyObject_BeforeStyleChange(Cancel As Boolean)
   If MsgBox("Line style about to be changed. Allow the change?", _
   vbYesNo) = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also AfterStyleChange event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeTextChange Event

Syntax **Private Sub** *DiagramObject*_**BeforeTextChange**(*Cancel* As Boolean)

Description The BeforeTextChange event occurs before a text change is processed for the specified DiagramObject object. The programmer can cancel the text change by setting the *Cancel* parameter to True.

Example The following example attempts to change the text on a shape, which fires the event. The user has the option to cancel the change.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   ' Create a shape
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Switch to the DiagramObject level
   Set igxDiagramObject = igxShape1.DiagramObject
   ' Set the text of the shape
   igxDiagramObject.Shape.Text = "Activity 1"
   MsgBox "Click OK to change the text."
   ' Change the text
   igxDiagramObject.Shape.Text = "Activity A"
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_BeforeTextChange(Cancel As Boolean)
   ' If the user answers "No", cancel the change
   If MsgBox("The shape is about to have it's text changed." _
   & FieldValue & Chr(13) & "Proceed with the change?", _
   vbYesNo) = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also AfterTextChange event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

BeforeUngroup Event

Syntax **Private Sub** *DiagramObject*_**BeforeUngroup**(*Group* As Group, *Cancel* As Boolean)

Description The BeforeUngroup event occurs before the DiagramObject is ungrouped. The Group parameter identifies the Group object for which the “ungroup” operation has been requested. The programmer can cancel the ungroup operation by setting the *Cancel* parameter to True.

Example The following example attempts to ungroup a shape, which fires the event. The user has the option to cancel ungroup operation.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxObjectRange As ObjectRange
   Dim igxGroup As Group
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 4, 1440)
   Set igxObjectRange = ActiveDiagram.MakeObjectRange
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to create the group."
   ' Add the shapes to the ObjectRange
   igxObjectRange.Add igxShape1.DiagramObject
   igxObjectRange.Add igxShape2.DiagramObject
   ' Make a Group from the ObjectRange
   Set igxGroup = igxObjectRange.Group
   MsgBox "Click OK to ungroup the objects."
   ' Ungroup the objects
   igxGroup.Ungroup
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_BeforeUngroup(ByVal Group As IXGroup, Cancel As
Boolean)
   ' If the user answers "No", cancel the change
   If MsgBox("The shape is about to be ungrouped." _
   & FieldValue & Chr(13) & "Proceed with ungrouping?", _
   vbYesNo) = vbNo Then
       Cancel = True
   End If
End Sub
```

See Also AfterUngroup event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Bottom Property

Syntax *DiagramObject*.**Bottom**

Data Type Long (read/write)

Description The Bottom property specifies the location of the bottom edge of the referenced DiagramObject object. The Bottom property and the Top property are mutually exclusive. The value of the one set most recently is applied. Values for this property are specified in twips (1440 twips = 1 inch).

The Top, Bottom, Left, Right, CenterX and CenterY properties all allow you to position an object on a diagram.

Example The following example moves the bottom of a shape down two inches.

```
' Dimension the variables
Dim igxShape As Shape
' Add a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "Click OK to move the shape down two inches."
' Move the bottom of the shape
igxShape.DiagramObject.Bottom = igxShape.DiagramObject.Bottom + 2880
' Pause for user
MsgBox "Click OK to continue."
```

See Also

CenterX property

CenterY property

Left property

Right property

Top property

Move method

Resize method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

CenterX Property

Syntax *DiagramObject*.**CenterX**

Data Type Long (read/write)

Description The CenterX property moves the specified DiagramObject object in the X direction (horizontally) so that the object's center is at the designated position in X. The value is specified in twips (1440 twips = 1 inch).

The CenterX and CenterY properties provides an alternative way of positioning objects based on their centers rather than their edges, which are used by the Left, Right, Top, and Bottom properties. The Top, Bottom, Left, Right, CenterX and CenterY properties all allow you to position an object on a diagram.

Example The following example moves the center of a shape two inches to the right.

```
' Dimension the variables
Dim igxShape As Shape
' Add a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "Click OK to move the shape's center two inches to the right."
' Move the shape
igxShape.DiagramObject.CenterX = igxShape.DiagramObject.CenterX + 2880
' Pause for user
MsgBox "Click OK to continue."
```

See Also

Bottom property

CenterY property

Left property

Right property

Top property

Move method

Resize method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

CenterY Property

Syntax *DiagramObject*.**CenterY**

Data Type Long (read/write)

Description The CenterY property moves the specified DiagramObject object in the Y direction (vertically) so that the object's center is at the designated position in Y. The value is specified in twips (1440 twips = 1 inch).

The CenterX and CenterY properties provides an alternative way of positioning objects based on their centers rather than their edges, which are used by the Left, Right, Top, and Bottom properties. The Top, Bottom, Left, Right, CenterX and CenterY properties all allow you to position an object on a diagram.

Example The following example moves a shape's center two inches down.

```
' Dimension the variables
Dim igxShape As Shape
' Add a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "Click OK to move the shape's center two inches down."
' Move the shape
igxShape.DiagramObject.CenterY = igxShape.DiagramObject.CenterY + 2880
' Pause for user
MsgBox "Click OK to continue."
```

See Also

Bottom property

CenterX property

Left property

Right property

Top property

Move method

Resize method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Close Event

Syntax **Private Sub** *DiagramObject*_**Close**()

Description The Close event occurs before the document that contains the specified DiagramObject object is closed.

Example The following example creates a DiagramObject that hears events. It then closes the document. This fires the Close event, which informs the user that the document that contains the shape is being closed.

```
' Dimension a variable that hears DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape As Shape
   ' Add a shape
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the DiagramObject
   Set igxDiagramObject = igxShape.DiagramObject
   ' Close the document
   MsgBox "Click OK to close the document."
   ActiveDocument.CloseDocument
End Sub

Private Sub igxDiagramObject_Close()
   MsgBox "The shape's document is closing."
End Sub
```

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

ConnectorLine Property

Syntax *DiagramObject*.**ConnectorLine**

Data Type ConnectorLine object (read-only, See Object Properties)

Description The ConnectorLine property returns the ConnectorLine object for the specified DiagramObject object, if the DiagramObject is of type ixObjectConnector (refer to the Type property). If the specified DiagramObject is not a ConnectorLine object, an error is returned.

Example The following example creates two shapes, and connects them with a connector line. It then accesses the ConnectorLine object to change the color of the line.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxDiagramObject As DiagramObject
' Add two shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Set the DiagramObject
Set igxDiagramObject = igxConnector.DiagramObject
' Change the color
MsgBox "Click OK to change the color of the connector line."
igxDiagramObject.ConnectorLine.LineColor = vbGreen
MsgBox "Click OK to continue."
```

See Also Type property

ConnectorLine object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

ContextMenu Event

Syntax **Private Sub** *DiagramObject*_**ContextMenu**(*CommandBar* As CommandBar)

Description The ContextMenu event occurs when the specified DiagramObject is right-clicked with the mouse, which opens the object's context menu. The *CommandBar* parameter contains the context menu object (a CommandBar object), which can be used to alter the appearance of the menu before it is displayed in the interface.

Example The following example demonstrates the ContextMenu event. To try it, put this code in a diagram code window, then go to the diagram and add a few shapes. Then, right-click on the shapes and observe the altered context menu. This example event alters the captions on the first three items in the context menu.

```
Private Sub AnyObject_ContextMenu(ByVal CommandBar As CommandBar)
   ' AnyObject refers to any DiagramObject on the diagram
   ' Alter the first three items on the context menu
   CommandBar.CommandBarItems.Item(1).Caption = "CUT!"
   CommandBar.CommandBarItems.Item(2).Caption = "COPY!"
   CommandBar.CommandBarItems.Item(3).Caption = "PASTE!"
End Sub
```

See Also CommandBar object

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

CreateVbaControl Method

Syntax *DiagramObject*.**CreateVbaControl**

Description The CreateVbaControl method makes a DiagramObject a VBA control. This causes the DiagramObject to exist in the Visual Basic environment as an object. The object can then be used to listen to DiagramObject events, for instance.

Example The following example demonstrates how a DiagramObject must be converted to a VBA control before it can hear events.

```
Private Sub Main()
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
' Create VBA controls for the two shapes
For Each DiagramObject In ActiveDiagram.DiagramObjects
   If (DiagramObject.IsVbaControl = False) Then
       DiagramObject.CreateVbaControl
   End If
Next DiagramObject
End Sub

Private Sub Shape2_BeforeClick(ByVal X As Double, ByVal Y As Double, Cancel As
Boolean)
   Shape2.InputConnectorLines(1).ConnectorLine.DestinationArrowColor _
      = RGB(Rnd(1) * 255, Rnd(1) * 255, Rnd(1) * 255)
End Sub
```

See Also DeleteVbaControl method

IsVbaControl property

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

CustomDataValues Property

Syntax *DiagramObject*.**CustomDataValues**

Data Type CustomDataValues collection object (read-only, See Object Properties)

Description The CustomDataValues property returns the CustomDataValues collection for the specified DiagramObject object.

Example The following example adds a CustomDataField to the document, and then changes the value in a shape.

```
' Dimension the variables
Dim igxShape As Shape
' Set a shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the diagram object variable
Set igxDiagramObject = igxShape.DiagramObject
' Add a CustomDataField to the document
ActiveDocument.CustomDataDefinitions.Add MyField, _
   ixCustomDataFormatTextBase
' Set the CustomData Text Field of the shape
igxDiagramObject.CustomDataValues.Item _
   (1, ixCustomDataText).Value = "My data"
' Change the CustomData Text Field of the shape
MsgBox "Click OK to change the CustomDataValue in the shape."
igxDiagramObject.CustomDataValues.Item _
   (1, ixCustomDataText).Value = "My other data"
MsgBox "Click OK to continue."
```

See Also CustomDataValues object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

DeleteObject Event

Syntax **Private Sub** *DiagramObject*_**DeleteObject**()

Description The DeleteObject event occurs when a DiagramObject object is deleted.

Example The following example deletes a shape, which triggers the event. The user is then informed that the shape is no longer available.

```
' Dimension a variable that listens to DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape As Shape
   ' Add a shape object
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape.DiagramObject
   MsgBox "Click OK to delete the shape."
   igxShape.DiagramObject.DeleteDiagramObject
End Sub

Private Sub igxDiagramObject_DeleteObject()
    MsgBox "The shape was deleted. It is no longer available."
End Sub
```

See Also BeforeDelete event

DeleteDiagramObject method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

DeleteDiagramObject Method

Syntax *DiagramObject*.**DeleteDiagramObject**

Description The DeleteDiagramObject method deletes the specified DiagramObject object.

Example The following example creates two shapes in the active diagram, and connects them with a connector line. Then the DiagramObjects collection is examined for any ConnectorLine objects. If one is found, it is deleted using the DeleteDiagramObject method.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
MsgBox "Click OK to delete any connector objects"
' Find any connectors and delete them
For Each DiagramObject In ActiveDiagram.DiagramObjects
    If (DiagramObject.Type = ixObjectConnector) Then
        DiagramObject.DeleteDiagramObject
    End If
Next DiagramObject
MsgBox "View the diagram"
```

See Also BeforeDelete event

DeleteObject event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

DeleteVbaControl Method

Syntax *DiagramObject*.**DeleteVbaControl**

Description The DeleteVbaControl method deletes an existing DiagramObject object as a VBA control. That is, if an object has been made a VBA control using the CreateVbaControl method, this method deletes the object as a VBA control; the method does not delete the object.

You can use the IsVbaControl property to determine whether an object is a VBA control.

Example The following example creates two shapes in the active diagram and connects them. It then creates a VBA control for each diagram object. Then it removes the VBA control for any ConnectorLine objects in the diagram.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
' Create VBA controls for the two shapes and connector line
For Each DiagramObject In ActiveDiagram.DiagramObjects
   If (DiagramObject.IsVbaControl = False) Then
       DiagramObject.CreateVbaControl
   End If
Next DiagramObject
' Delete the VBA Control for any connector line object
For Each DiagramObject In ActiveDiagram.DiagramObjects
   If (DiagramObject.Type = ixObjectConnector) Then
       DiagramObject.DeleteVbaControl
   End If
Next DiagramObject
```

See Also CreateVbaControl method

IsVbaControl property

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Department Property

Syntax *DiagramObject*.**Department**

Data Type Department object (read-only, See Object Properties)

Description The Department property returns the Department object for the specified DiagramObject object, if the DiagramObject is of type ixObjectDepartment (refer to the Type property). If the specified DiagramObject is not a Department object, an error is returned.

Example The following example creates two shapes, and connects them with a connector line. It then adds two departments. Using the DiagramObjects collection, it displays all the DiagramObject object in the diagram, and their type. After that, it looks through the collection for any Department objects, and if found, changes the Process area of the department to cyan.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxDiagramObject As DiagramObject
Dim sList As String
' Add two shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add two departments to the diagram
ActiveDiagram.Departments.AddDepartment ("Dept. 1")
ActiveDiagram.Departments.AddDepartment ("Dept. 2")
MsgBox "View the diagram"
' List the DiagramObjects in the diagram and their type
sList = ""
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
    Select Case ActiveDiagram.DiagramObjects.Item(iCount).Type
        Case ixObjectShape:
            sList = sList & "Item " & iCount & ": " _
                & "A Shape object" & Chr(13)
        Case ixObjectDepartment:
            sList = sList & "Item " & iCount & ": " _
                & "A Department object" & Chr(13)
        Case ixObjectConnector:
            sList = sList & "Item " & iCount & ": " _
                & "A ConnectorLine object" & Chr(13)
    End Select
Next iCount
MsgBox "The DiagramObjects collection contains: " _
        & Chr(13) & sList
' For any Department object, change its Process area fill
' format color to Cyan
For Each igxDiagramObject In ActiveDiagram.DiagramObjects
    If (igxDiagramObject.Type = ixObjectDepartment) Then
        igxDiagramObject.Department.ProcessFillFormat _
            .FillColor = vbCyan
    MsgBox "Department object found. Click OK to continue."
```

```
        End If
    Next igxDiagramObject
```

See Also

Type property

Department object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Deselect Event

Syntax **Private Sub** *DiagramObject*_**Deselect**()

Description The DeselectEvent occurs when the specified DiagramObject is deselected.

Example The following example creates a shape in the active diagram, and then selects the shape. Next the shape is deselected, which fires the event.

```
' Dimension a module variable that hears DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject


Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   ' Create a shape
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape1.DiagramObject
   MsgBox "Click OK to select the shape."
   ' Select the shape
   ActiveDiagram.Selection.Add igxDiagramObject
   MsgBox "Click OK to deselect the object."
   ' Deselect the object
   ActiveDiagram.Selection.Remove igxDiagramObject
   ' Pause for the user
   MsgBox "Click OK to continue."
End Sub

Private Sub igxDiagramObject_Deselect()
   MsgBox "The DiagramObject has been deselected."
End Sub
```

See Also BeforeSelect event

Select event

Selected property

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Diagram Property

Syntax *DiagramObject*.**Diagram**

Data Type Diagram object (read-only, See Object Properties)

Description The Diagram property returns the Diagram object that contains the specified DiagramObject object.

Example The following example creates a shape, and the displays the name of the diagram in which it resides.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDiagramObject As DiagramObject
' Create a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the diagram object variable
Set igxDiagramObject = igxShape1.DiagramObject
' Display the name of the shape's diagram
MsgBox "This shape resides on " & igxDiagramObject.Diagram.Name
```

See Also Diagram object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Fields Property

Syntax *DiagramObject*.**Fields**

Data Type Fields collection object (read-only, See Object Properties)

Description The Fields property returns the Fields collection for the specified DiagramObject object.

Example The following example creates a shape, and then adds a field to the shape which displays the shape's creation date.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDiagramObject As DiagramObject
' Create a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the diagram object variable
Set igxDiagramObject = igxShape1.DiagramObject
' Add a date field to the shape
igxDiagramObject.Fields.Add ixFieldTextCreateDate, Now, ixFieldBelow
' Pause for user
MsgBox "Date field added to the shape."
```

See Also Field object

Fields object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

FireUserEvent Method

Syntax *DiagramObject*.**FireUserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The FireUserEvent method fires the "UserEvent" for the specified document. You can use this functionality to send messages to any DiagramObject object that is listening to events.

You must specify an *EventIdentifier* argument (a string) to use for your event. You might choose to use something like your company name followed by the event name. You should choose a name that won't conflict with names picked by other developers.

You can pass one parameter to the event (the *Parameter* argument). This parameter is a Variant, so one logical choice is to pass a Class.

Then, you can write code in a UserEvent handler to perform some actions when your event fires. This code should be of the form:

```
If EventIdentifier = "<<Your identifier string>>" Then
   << Write your code here >>
End If
```

Example The following example extends the BeforeClick event using a UserEvent. The BeforeClick event fires a "RightSideClicked" user event, which fires if the user clicks toward the right side of a shape.

```
' Dimension a variable that hears DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

' Run this subroutine to test the event
Private Sub Main()
   Dim igxShape As Shape
   ' Add a shape
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxDiagramObject = igxShape.DiagramObject
   MsgBox "The shape and events are ready. Return to the " _
       "diagram and try" & Chr(13) _
       & "clicking the right side of the shape."
End Sub

' The BeforeClick event
Private Sub igxDiagramObject_BeforeClick(ByVal X As Double, ByVal Y As Double,
Cancel As Boolean)
   igxDiagramObject.FireUserEvent "RightSideClick", X
End Sub

' This event handler runs every time any FireUserEvent method
' is used in the system
Private Sub igxDiagramObject_UserEvent(ByVal EventIdentifier As String, ByVal
Parameter As Variant)
   ' Check if the Identifier string is the one we want
   If EventIdentifier = "RightSideClick" Then
       ' Custom click parameter
       If Parameter > 0.6 Then
           MsgBox "The right side of the shape was clicked."
       End If
   End If
End Sub
```

See Also UserEvent event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

GetInterface Event

Syntax **Private Sub** *DiagramObject*_**GetInterface**(ByVal *TypeName* As String, *Interface* As Object)

Description The GetInterface event occurs when the DiagramObject.AsType property is used. The AsType property allows you to add your own properties and methods to a DiagramObject object, extending the object model. The properties and methods can be organized by using unique type names.

The *TypeName* argument is a string that distinguishes the custom type. It can be any string the programmer chooses, but it must be unique within the environment. In an integrated environment, other programmers may be accessing the DiagramObject object, and using it's AsType property. To prevent conflicting type names, it is suggested that you use your company or department name, followed by a descriptive type name (for example, "MyCompanyFactory").

Use the following basic steps to implement a custom property or method for the DiagramObject object.

1. Use DiagramObject.AsType ("my type name").MyMethod in your code.
2. Create a new Class, and design properties and methods in the class.
3. Set up the GetInterface event to check the TypeName string passed to it. If it matches your type name, set the Interface parameter equal to your new class.

When you use DiagramObject.AsType(*TypeName*) in your code, you gain access to the properties and methods that you have defined in the new Class. The DiagramObject.AsType property automatically fires an event called GetInterface. The GetInterface event can have one or more AsType's defined, each one distinguished by a unique type name. Based on the type name, the GetInterface event redirects execution to your new Class by setting the Interface parameter. If the Interface parameter is set to your new Class, the Class properties and methods become exposed to the DiagramObject object.

Notes When you extend an iGrafx Professional object using the GetInterface event, you need to keep in mind that other developers may be using this event also. To be a good citizen, you should do the following:

- Be sure to pick a name that is likely to be unique for your AsType name. In the example above, "MyType" is too generic and it is possible that another developer could use the same name. Instead, follow the convention of using your name or your company name, a period, and a description of the type. For example, if you were writing a type that extended Application to add additional internet capabilities, and your company name was "Micrografx", you could name your AsType name "Micrografx.InternetExtension".
- When you write code in the GetInterface event, keep it simple. You should not do any time consuming operation in the GetInterface event such as querying a database or displaying a dialog box.
- When you write code in the GetInterface event, be aware of the current state of the Interface parameter. In the example above, this is illustrated by the code fragment "Interface Is Nothing". If this code fragment evaluates to true, then it is safe to Set the interface to your class. If this code fragment evaluates to false then someone else has already responded to the event and set the interface to their class. If this condition arises, you should try changing your AsType name.

Example Using the AsType property, the GetInterface event, and VBA's support for Classes, you can extend key iGrafx objects. The following example creates a shape AsType Airplane, and retreives it's AircraftType property.

The first step to doing this is creating a VBA class. The following code creates a simple class which has two properties—AirCraftType, and Departure.

Insert a new class called Class1, and copy this block of code into it.

```
' Class
```

```
Public Property Get AirCraftType() As String
   AirCraftType = "Boeing 747"
End Property

Public Property Get Departure() As String
   Departure = "Monday, 8:05 AM"
End Property
```

The following two blocks of code go in the project code window above the new Class1.

```
' Run this to test the event
Private Sub Main()
   Dim igxShape As Shape
   Dim igxDiagramObject As DiagramObject
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxDiagramObject = igxShape.DiagramObject
   MsgBox "The aircraft is a " _
        & igxDiagramObject.AsType("Airplane").AirCraftType
End Sub

' The GetInterface event is fired whenever the AsType method is used.
' Based on the TypeName, redirect the interface to your custom class
Private Sub AnyObject_GetInterface(ByVal TypeName As String, Interface As
Object)
   ' If the broadcast type name is "Airplane", then set the interface
   If TypeName = "Airplane" Then
      ' TypeName gets broadcast everywhere, so we need to check if
      ' something else grabbed and set the Interface first
      If Interface Is Nothing Then
          Set Interface = New Class1
      Else
          MsgBox "ERROR: Someone else is using Airplane AsType"
      End If
   End If
End Sub
```

See Also AsType property

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Group Property

Syntax *DiagramObject*.**Group**

Data Type Group object (read-only, See Object Properties)

Description The Group property returns a Group object if the specified DiagramObject is a Group object.

Error The Group property returns a Group object only if the DiagramObject is of type *ixObjectGroup* (refer to the Type property). If the specified DiagramObject is not a Group object, the property returns an invalid Group object. Subsequent attempts to use the invalid Group object produce a runtime error.

Example The following example creates two shapes in the active diagram, and then makes a group with those shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector As ConnectorLine
Dim igxDiagramObject As DiagramObject
Dim igxObjRange As ObjectRange
Dim igxGroup As Group
' Add two shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 3)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add the shapes to the ObjectRange
MsgBox "Click OK to add the shapes to the ObjectRange."
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
' Create a group that contains the two shapes
MsgBox "Click OK to group the objects"
Set igxGroup = igxObjectRange.Group
' Specify a name for the Group object
igxGroup.DiagramObject.ObjectName = "Group 1"
' Display the Group Object name using the DiagramObject property
MsgBox "The Group object name is: " _
   & igxGroup.DiagramObject.ObjectName
MsgBox "The group contains " & igxGroup.ObjectRange.Count & " items"
' Test whether the shapes in the diagram are members of a group
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
    Set igxDiagramObject = ActiveDiagram.DiagramObjects.Item(iCount)
    If (igxDiagramObject.Type = ixObjectShape) Then
        If (igxDiagramObject.IsGrouped) Then
            MsgBox "The shape is not part of a group"
        Else
            MsgBox "The shape is a member of a group"
```

```
        End If
    End If
Next iCount
MsgBox "There are " & ActiveDiagram.DiagramObjects.Count _
    & " diagram objects"
' List the DiagramObjects in the diagram and their type
sList = ""
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
    Select Case ActiveDiagram.DiagramObjects.Item(iCount).Type
        Case ixObjectShape:
            sList = sList & "Item " & iCount & ": " _
                & "A Shape object" & Chr(13)
        Case ixObjectDepartment:
            sList = sList & "Item " & iCount & ": " _
                & "A Department object" & Chr(13)
        Case ixObjectConnector:
            sList = sList & "Item " & iCount & ": " _
                & "A ConnectorLine object" & Chr(13)
        Case ixObjectGroup:
            sList = sList & "Item " & iCount & ": " _
                & "A Group object" & Chr(13)
    End Select
Next iCount
MsgBox "The DiagramObjects collection contains: " _
        & Chr(13) & sList
' For any Department object, change its Process area fill
' format color to Cyan
For Each igxDiagramObject In ActiveDiagram.DiagramObjects
    If (igxDiagramObject.Type = ixObjectDepartment) Then
        igxDiagramObject.Department.ProcessFillFormat _
            .FillColor = vbCyan
    MsgBox "Department object found. Click OK to continue."
    End If
Next igxDiagramObject
```

See Also

Type property

Group object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

ID Property

Syntax *DiagramObject*.**ID**

Data Type Long (read-only)

Description The ID property gets the unique ID that iGrafx Professional assigns to the specified DiagramObject object when it is created.

Example The following example displays the IDs of shape objects in the diagram.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxTypeName As String
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Create VBA controls for the two shapes
For Each igxDiagramObject In ActiveDiagram.DiagramObjects
    If (igxDiagramObject.Type = ixShape) Then
        igxTypeName = "Shape object"
        MsgBox ("The " & igxDiagramObject.ObjectName & _
            " " & igxTypeName & " has an ID of " & _
            igxDiagramObject.ID)
    ElseIf (igxDiagramObject.Type = ixConnector) Then
        igxTypeName = "Connector object"
        MsgBox ("The " & igxDiagramObject.ObjectName & _
            " " & igxTypeName & " has an ID of " & _
            igxDiagramObject.ID)
    End If
Next igxDiagramObject
```

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

IsGrouped Property

Syntax *DiagramObject*.**IsGrouped**[= **{True | False}**]

Data Type Boolean (read-only)

Description The IsGrouped property indicates whether the specified DiagramObject is a member of a group. Use this property to determine whether a DiagramObject is in a group before accessing the group through the Group property.

Example Refer to the example for the Group property.

See Also Group property

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

IsVBAControl Property

Syntax *DiagramObject*.**IsVBAControl**[= **{True | False}**]

Data Type Boolean (read-only)

Description The IsVBAControl property is used to test whether the specified DiagramObject object is a VBA control. To refer to an object directly in your code, for instance, an event procedure, the object must be a VBA control. You can make an object a VBA control with the CreateVbaControl method.

Example The following example creates two shapes in the active diagram and connects them. It then turns each DiagramObject into a VBA control by using the IsVBAControl property to first test whether the DiagramObject is already a VBA control.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Create VBA controls for the two shapes
For Each DiagramObject In ActiveDiagram.DiagramObjects
    If (DiagramObject.IsVbaControl = False) Then
       If (DiagramObject.Type = ixShape) Then
           DiagramObject.CreateVbaControl
           igxTypeName = "Shape object"
           MsgBox (DiagramObject.ObjectName & " is a " & igxTypeName _
               & " and is now a VBA control")
       End If
    End If
Next DiagramObject
```

Once you have created VBA controls for your shapes, you can write procedures or subroutines specifically for each shape. For example, you could write a BeforeClick procedure specifically for Shape 2.

```
Private Sub Shape2_BeforeClick(ByVal X As Double, ByVal Y As Double, Cancel As
Boolean)
   Shape2.InputConnectorLines(1).ConnectorLine.LineColor _
       = RGB(Rnd(1) * 255, Rnd(1) * 255, Rnd(1) * 255)
End Sub
```

See Also

CreateVbaControl method

DeleteVbaControl method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Layer Property

Syntax *DiagramObject*.**Layer**

Data Type Layer object (read-only, See Object Properties)

Description The Layer property returns the Layer object on which the specified DiagramObject resides.

Example The following example creates a shape, and displays the name of the layer it is on.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
' Add a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the shape's DiagramObject
Set igxDiagramObject = igxShape.DiagramObject
' Display the shape's layer name
MsgBox "The shape resides on " & igxDiagramObject.Layer.Name
```

See Also Layer object

Layers object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Left Property

Syntax *DiagramObject*.**Left**

Data Type Long (read/write)

Description The Left property specifies the location of the left side of the referenced DiagramObject object. The Left property and the Right property are mutually exclusive. The value of the one set most recently is applied. Values for this property are specified in twips (1440 twips = 1 inch).

Example The following example creates two shapes in the active diagram at the same location, and sets the fill of the second shape to ixFillNone. It then resizes the second shape to be 2 inches by 3 inches. Then, using the Left and Right properties, it moves the second shape.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape at the same position
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape2.FillType = ixFillNone
' Resize the second shape to make it 2 inches by 3 inches
ActiveDiagram.DiagramObjects.Item(2).Resize 1440 * 2, 1440 * 3
' Sequentially set the Left, then the Right properties
' to move the shape
ActiveDiagram.DiagramObjects.Item(2).Left = 1440 * 2
MsgBox "Left side of shape at 2 inch mark"
ActiveDiagram.DiagramObjects.Item(2).Right = 1440 * 5
MsgBox "Right side of shape is at 5 inch mark"
```

See Also

Bottom property

CenterX property

CenterY property

Right property

Top property

Move method

Resize method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Legend Property

Syntax *DiagramObject*.**Legend**

Data Type Legend object (read-only, See Object Properties)

Description The Legend property returns a Legend object if the specified DiagramObject is a legend (Type property = ixObjectLegend). When the DiagramObject is a legend, this property provides access to the properties, methods, and events for controlling a Legend object.

Use the Type property to determine if a DiagramObject is a Legend object. If the DiagramObject is not a Legend object, then this property returns the 'Nothing' value.

Example The following example creates two connected shapes, a Graphic object, and a TextGraphicObject in the active diagram. It then creates a Legend object, and then searches through the DiagramObjects collection to find the legend. Once found, the formatting properties of the Legend object are set.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxGraphic As TextGraphicObject
Dim igxTextObj As TextGraphicObject
Dim igxLegend As Legend
Dim igxGraphicBuilder As New GraphicBuilder
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right of Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "View the diagram"
' Draw a direct connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
   (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
   ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
   ixDirWest, ixConnectRelativeToShape)
MsgBox "View the diagram"
' Create a graphic consisting of a rectangle and an ellipse
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
   FillFormat.FillColor = vbBlue
' Add the graphic in the active diagram
Set igxGraphic = ActiveDiagram.DiagramObjects. _
   AddGraphic(igxGraphicBuilder.Graphic, 1440, 2880, 1440, 1440)
MsgBox "View the diagram"
' Create a text object in the active diagram
Set igxTextObj = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440 * 4, Width:=1440, _
   Text:="I am a Text Object")
MsgBox "View the diagram"
' Create a Legend object in the active diagram
Set igxLegend = ActiveDiagram.DiagramObjects. _
```

```
   AddLegend(1440 * 5, 1440 * 6)
MsgBox "View the diagram"
' Find any DiagramObjects that are of type Legend
For Each DiagramObject In DiagramObjects
   If (DiagramObject.Type = ixObjectLegend) Then
       MsgBox "Legend object found"
       ' Set formatting properties for the Legend
       With igxLegend
           .FillFormat.FillType = ixFillGradient
           .FillFormat.GradientFormat.Type = ixGradientSquare
           .FillFormat.FillColor = vbBlue
           .FillFormat.BackColor = vbYellow
           .LineFormat.Style = ixLineNormal
           .LineFormat.Width = 3
           .LineFormat.Color = vbBlack
           .Font.Name = "Arial"
           .Font.Bold = True
           .Font.Size = 12
       End With
       MsgBox "Legend formatting done"
   End If
Next DiagramObject
```

See Also

Type property

Legend object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Load Event

Syntax **Private Sub** *DiagramObject*_**Load**()

Description The Load event occurs for the specified DiagramObject when the document containing the DiagramObject is loaded. This event is useful if you want to do something with or to a particular object as soon as the document is loaded.

Example The following example causes a shape to be automatically selected every time the document is loaded.

```
Private Sub Shape1_Load()
    ActiveDiagram.Selection.Add Shape1.DiagramObject
End Sub
```

To try this example:

1. Create a shape called Shape1
2. Put the above event code in diagram's code window
3. Save the document
4. Close the document
5. Load the document

The shape is selected as soon as the document is loaded.

See Also Close event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

MakeObjectRange Method

Syntax *DiagramObject*.**MakeObjectRange** As ObjectRange

Description The MakeObjectRange method creates an ObjectRange object that contains the specified DiagramObject object.

The MakeObjectRange method creates an ObjectRange object that contains the specified DiagramObject object. The MakeObjectRange method is used to create a new ObjectRange object. A new ObjectRange object cannot be created using the New keyword, or using Add methods; the MakeObjectRange method must be used.

Example The following example creates two shapes. It then makes an ObjectRange object and adds the shapes to it. Finally it changes the colors of any objects in the ObjectRange.

```
' Dimension the variables
Dim igxDiagramObject As DiagramObject
Dim igxObjectRange As ObjectRange
Dim igxShape1 As Shape
Dim igxShape2 As Shape
' Add two shapes to the diagram
MsgBox "Click OK to add two shapes to the diagram."
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
' Get the shape's DiagramObject
Set igxDiagramObject = igxShape1.DiagramObject
' Make an ObjectRange
Set igxObjectRange = igxDiagramObject.MakeObjectRange
' Add the shapes to the ObjectRange
MsgBox "Click OK to add the shapes to the ObjectRange."
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
' Change the fill color of the ObjectRange to blue
MsgBox "Now click OK to change the ObjectRange to blue."
igxObjectRange.FillFormat.FillColor = vbBlue
MsgBox "Click OK to continue."
```

See Also ObjectRange object

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Modify Event

Syntax **Private Sub** *DiagramObject*_**Modify**()

Description The Modify event occurs when the specified DiagramObject object is modified. The DiagramObject can be a specific one that you have turned into a VBA control, a DiagramObject variable declared with the "WithEvents" keyword, or it can be an “AnyObject” object (see the AnyControls object).

A modification or change to any property of the specified diagram object triggers this event. Therefore, the event can be useful for protecting certain objects from being changed by a user, or for tracking when changes occur.

Warning **Do not use this event to change anything on a diagram.** If the event adds or deletes a DiagramObject, or changes any aspect of a DiagramObject, the change fires the event again, creating a recursion loop which hangs the system. Use the event to display a message box, set flags, or track changes. Do not change any aspect of any DiagramObject objects from within this event.

Though the event fires reliably, the Modify event may fire twice or more for what appears to be a single modification to a DiagramObject. This behavior is not entirely predictable due to the event's ties to the system at the core level. If greater precision is required, try using any of the "Before" events, which provide precise monitoring of individual changes to DiagramObject objects.

Example The following example sets up the Modify event with the AnyObject DiagarmObject object. The event displays the date and time of modifications.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents AnyDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxFont As Font
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
   ' Set the diagram object variable to AnyObject
   Set AnyDiagramObject = ActiveDiagram.AnyControls.AnyObject
   ' Pause for the user
   MsgBox "The event is ready.  Try modifying diagram objects."
End Sub

Private Sub AnyDiagramObject_Modify()
   MsgBox "The diagram was last modified " & Now
End Sub
```

See Also CreateVbaControl method

AnyControls object

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Move Method

Syntax *DiagramObject*.**Move** (*Left* As Long, *Top* As Long, [*Width* As Long = -1], [*Height* As Long = -1], [*Angle* As Double])

Description The Move method allows you to move a DiagramObject object to a different location on the same diagram. Optionally, this method also allows you to resize and rotate the DiagramObject object. Moving the object is not required to use the resizing or rotation arguments—see the Example section.

The *Left*, *Top*, *Width*, and *Height* arguments are specified in units of twips (1440 twips = 1 inch). The *Angle* argument is specified in units of degrees of rotation.

Example The following example creates five shapes at the same location in the active diagram. Using the Move method, the shapes are spaced out in a horizontal row half an inch apart, and filled with a random color. The Move method is used again to relocate the shapes and resize them. Finally, the Move method is used again to adjust the location of the shapes and to rotate them counterclockwise progressively in increments of 15 degrees.

```
' Dimension the variables
Dim igxShape As Shape
Dim iCount As Integer
Dim iSpacing As Integer
' Create five shapes in the same location on the active diagram
For iCount = 1 To 5
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
Next iCount
MsgBox "View state of the diagram"
' Arrange the 5 shapes in a horizontal row spaced half an inch
' apart using the Move method and fill them with a random color
iSpacing = 1
For Each DiagramObject In ActiveDiagram.DiagramObjects
   If (DiagramObject.Type = 0) Then
       Call DiagramObject.Move(720 * iSpacing, 1440)
       DiagramObject.Shape.FillType = ixFillSolid
       DiagramObject.Shape.FillColor = RGB(Rnd(1) * 255, _
           Rnd(1) * 255, Rnd(1) * 255)
       iSpacing = iSpacing + 3
   End If
Next DiagramObject
MsgBox "View state of the diagram"
iSpacing = 1
' Move each shape to the left 1/8 inch and up half an inch, and
' resize to 3/4 inch wide and progressively taller starting at
' half an inch
For Each DiagramObject In ActiveDiagram.DiagramObjects
   If (DiagramObject.Type = 0) Then
       Call DiagramObject.Move(DiagramObject.Left - 180, _
           DiagramObject.Top - 720, 1080, 720 * iSpacing)
       iSpacing = iSpacing + 2
   End If
   MsgBox "View the diagram"
Next DiagramObject
iSpacing = 1
' Use the Move method to relocate the shapes 1/16 inch to the
```

```
' right and down 1/4 inch, and rotate each shape counterclockwise
' progressively in 15 degree increments
For Each DiagramObject In ActiveDiagram.DiagramObjects
    If (DiagramObject.Type = 0) Then
        Call DiagramObject.Move(DiagramObject.Left + 90, _
            DiagramObject.Top + 360, , , -15 * iSpacing)
        iSpacing = iSpacing + 1
   End If
   MsgBox "View the diagram"
Next DiagramObject
```

See Also

Bottom property

CenterX property

CenterY property

Left property

Right property

Top property

Resize method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Name Property

Syntax *DiagramObject*.**Name**

Data Type String (read/write)

Description The Name property returns the string "DiagramObject". Writing values to this property has no effect.

This property is included only as a requirement of the Common Object Model used by Visual Basic. For a more useful naming property, see the ObjectName property.

Example The following example creates five shapes in the active diagram. It then displays the Name property for each of the five shapes, which is “DiagramObject”. It then attempts to change the Name property’s value for each shape to “Shape1”, “Shape2”, etc. However, displaying the Name property again shows that trying to change the property’s value has no effect.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDiagramObj As DiagramObject
Dim iCount As Integer
Dim iSpacing As Integer
' Create five shapes in the same location on the active diagram
For iCount = 1 To 5
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
Next iCount
MsgBox "View state of the diagram"
' Arrange the 5 shapes in a horizontal row spaced half an inch
' apart using the Move method and fill them with a random color
iSpacing = 1
For Each DiagramObject In ActiveDiagram.DiagramObjects
   If (DiagramObject.Type = 0) Then
        Call DiagramObject.Move(720 * iSpacing, 1440)
       DiagramObject.Shape.FillType = ixFillSolid
       DiagramObject.Shape.FillColor = RGB(Rnd(1) * 255, _
            Rnd(1) * 255, Rnd(1) * 255)
       iSpacing = iSpacing + 3
   End If
Next DiagramObject
MsgBox "View state of the diagram"
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
    Set igxDiagramObj = ActiveDiagram.DiagramObjects.Item(iCount)
    MsgBox "Item " & iCount & "'s Name property is: " _
        & igxDiagramObj.Name
    ' Change the name property
    igxDiagramObj.Name = "Shape" & Str(iCount)
    MsgBox "Tried to change the object's Name property: " _
        & "Did it work?" & Chr(13) & Chr(13) _
        & "Item " & iCount & "'s Name property is now: " _
        & igxDiagramObj.Name
Next iCount
```

See Also ObjectName property

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

New Event

Syntax **Private Sub** *DiagramObject*_**New**()

Description The New event occurs when a new DiagramObject object is added to a diagram. This event allows you to monitor the act of adding diagram objects to a diagram, and to initiate any actions necessary for your application.

Typically, this event is monitored through one of the AnyControl objects (AnyShape, AnyConnector, etc.).

Example The following example sets up a module variable that hears events from AnyObject on the diagram. It then adds shapes to the diagram, which fires the event. The event displays the number of objects in the diagarm each time one is added.

```
' Dimension a module variable that hears
' DiagramObject events
Private WithEvents AnyDiagramObject As DiagramObject

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Set the module variable to the AnyObject object
   Set AnyDiagramObject = ActiveDiagram.AnyControls.AnyObject
   MsgBox "Click OK to add shapes to the diagram."
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
End Sub

Private Sub AnyDiagramObject_New()
   MsgBox "There are now " & ActiveDiagram.DiagramObjects.Count & _
       " objects on the diagram."
End Sub
```

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Object Property

Syntax *DiagramObject*.**Object**

Data Type Object object (read-only, See Object Properties)

Description The Object property is included only as a requirement of the Common Object Model used by VisualBasic. It serves no purpose in VBA programs. This property returns Nothing.

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

ObjectName Property

Syntax *DiagramObject*.**ObjectName**

Data Type String (read/write)

Description The ObjectName property specifies the name of the DiagramObject object. For Shape objects, this property is set automatically by iGrafx Professional to the name of the shape type (for example, “Process” or “Decision”). For other diagram object types, iGrafx Professional does not automatically set this property value.

The ObjectName property can be used for any purpose you want, such as identifying specific objects or a group of similar objects. For Shape objects, the ObjectName property has a relationship with the ShapeClass.Name property, and somewhat more loosely with the ShapeLibraryItem.ToolTip property (as is demonstrated in the example). Refer to these other shape-related properties for information about their purpose and use.

A related topic is what name does an object get assigned when you turn it into a VBA control (see the CreateVbaControl method). For all diagram objects, the name you need to use when referring to the object as a VBA control is specified in the DiagramObject.Name property (refer to this topic for information).

Example The following example creates five objects in the active diagram: two shapes, one connector line, and two text-graphic objects. It then gets the ObjectName property for each diagram object and displays the names, illustrating that non-shape diagram object do not automatically have a string assigned to the ObjectName property. For shapes, the code then shows that three different properties (ShapeClass.Name, ShapeLibraryItem.ToolTip, and DiagramObject.ObjectName) all contain the same string. Next, the ObjectName property is set for all non-shape objects, and then the ObjectName property of the two shapes are changed. Finally, a message is displayed listing the values of the various properties that contain names for the shapes, and the ObjectName property for the non-shape objects.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxGraphic As TextGraphicObject
Dim igxTextObj As TextGraphicObject
Dim igxGraphicBuilder As New GraphicBuilder
Dim iCount As Integer
Dim igxObjList As String
Dim igxConnType As String
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right of Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "View the state of the diagram"
' Draw a direct connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
   ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
   ixDirWest, ixConnectRelativeToShape)
MsgBox "View the state of the diagram"
' Create a graphic consisting of a rectangle and an ellipse
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
```

```
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
   FillFormat.FillColor = vbBlue
' Add the graphic in the active diagram
Set igxGraphic = ActiveDiagram.DiagramObjects. _
   AddGraphic(igxGraphicBuilder.Graphic, 1440, 2880, 1440, 1440)
MsgBox "View the state of the diagram"
' Create a text object in the active diagram
Set igxTextObj = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440 * 4, Width:=1440, _
   Text:="I am a Text Object")
MsgBox "View the diagram. Next list the Object names of all objects."
' Create a list of the ObjectName property of each diagram object
For Each DiagramObject In DiagramObjects
   If (DiagramObject.ObjectName <> "") Then
       igxObjList = igxObjList & Chr$(13) _
           & DiagramObject.ObjectName
   Else
       igxObjList = igxObjList & Chr$(13) & "No object name set"
   End If
Next DiagramObject
' Display the list of object names collected from the
' ObjectName property of each of the diagram objects
MsgBox "The object names are: " & Chr$(13) & igxObjList
' Display the Name property from each shape's ShapeClass object
MsgBox "Shape1's ShapeClass Name property is: " _
   & igxShape1.ShapeClass.Name & Chr$(13) & _
   "Shape2's ShapeClass Name property is: " _
   & igxShape2.ShapeClass.Name
' Display the ToolTip property of both shapes
MsgBox "The ToolTip string for Shape1 is: " _
   & Application.ShapeLibraries.Item(1).Item(1).ToolTip _
   & Chr$(13) & "The ToolTip string for Shape2 is: " _
   & Application.ShapeLibraries.Item(1).Item(4).ToolTip
' Set the ObjectName property for the non-shape objects
For Each DiagramObject In DiagramObjects
   If (DiagramObject.ObjectName = "") Then
       Select Case DiagramObject.Type
           Case ixObjectTextGraphic:
               DiagramObject.ObjectName = "TextGraphicObject"
           Case ixObjectConnector:
               Select Case DiagramObject.ConnectorLine.Routing
                   Case ixRouteDirect:
                       igxConnType = "Direct Connector"
                   Case ixRouteRightAngle:
                       igxConnType = "RightAngle Connector"
                   Case ixRouteCurved:
                       igxConnType = "Curved Connector"
                   Case ixRouteOrgChart:
                       igxConnType = "OrgChart Connector"
                   Case ixRouteCauseAndEffect:
                       igxConnType = "Cause/Effect Connector"
                   Case ixRouteLightningBolt:
                       igxConnType = "LightningBolt Connector"
               End Select
               DiagramObject.ObjectName = igxConnType
```

```
        End Select
    End If
Next DiagramObject
' Change the ObjectName property for both shapes
igxShape1.DiagramObject.ObjectName = "Verification Process"
igxShape2.DiagramObject.ObjectName = "Release Verified Item"
' Clear the Object List string variable
igxObjList = ""
' Get the ObjectName property for all diagram objects
For Each DiagramObject In DiagramObjects
    If (DiagramObject.ObjectName <> "") Then
        igxObjList = igxObjList & Chr$(13) _
            & DiagramObject.ObjectName
    Else
        igxObjList = igxObjList & Chr$(13) & "No object name set"
    End If
Next DiagramObject
' Display the ShapeClass names and the ToolTip names for the
' shapes, followed by the list of object names collected from the
' ObjectName property of each of the diagram objects
MsgBox "Shape1's ShapeClass Name property is: " _
    & igxShape1.ShapeClass.Name & Chr$(13) & _
    "Shape2's ShapeClass Name property is: " _
    & igxShape2.ShapeClass.Name & Chr$(13) _
    & "The ToolTip string for Shape1 is: " _
    & Application.ShapeLibraries.Item(1).Item(1).ToolTip _
    & Chr$(13) & "The ToolTip string for Shape2 is: " _
    & Application.ShapeLibraries.Item(1).Item(4).ToolTip _
    & Chr$(13) & "The object names are: " & Chr$(13) & igxObjList
```

See Also

Name property

ShapeClass.Name property

ShapeLibraryItem.ToolTip property

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

OleObject Property

Syntax *DiagramObject*.**OleObject**

Data Type OleObject object (read-only, See Object Properties)

Description The OleObject property returns an OleObject object if the specified DiagramObject is an OleObject (Type property = ixObjectOle). When the DiagramObject is an OleObject, this property provides access to the properties, methods, and events for controlling an OleObject object.

Use the Type property to determine if a DiagramObject is an OleObject object. If the DiagramObject is not an OleObject object, then this property returns the 'Nothing' value.

Example The following example adds a Word document to the diagram as an OleObject. It then performs the –1 (Open) verb on the OleObject (standard Ole verbs are numbered, and use negative numbers.) This opens the document for editing it's contents while on the diagram.

```
' This example requires a sample Word document. You need to
' create one called C:\My Documents\Sample.doc
'
' Dimension the variables
Dim igxOleObject As OleObject
' Add an OLE object to the diagram
Set igxOleObject = ActiveDiagram.DiagramObjects.AddOleObject _
    ("C:\My Documents\sample.doc", 1100, 0, False, False)
' Display the OLE object's classname
MsgBox "A " & igxOleObject.ClassName & _
   " has been added to the diagram."
MsgBox "Click OK to do the Activate verb to editing the OleObject."
' "-1" is the Ole Activate verb which will open the document
' for editing
igxOleObject.DoVerb -1
```

See Also OleObject object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

PermanentDiagramObject Property

Syntax *DiagramObject*.**PermanentDiagramObject**

Data Type DiagramObject object (read-only, See Object Properties)

Description The PermanentDiagramObject property returns a DiagramObject object. The purpose of this property is to provide a means of holding on to the object an AnyControl is pointing at after an event is over.

The AnyControl objects are special VBA controls that are only valid during an event; these objects dynamically point at the "active" object that is firing the event. The PermanentDiagramObject property is used to “grab” the specific object the AnyControl is pointing at so that it can be used (or accessed) once the event is over.

As an example, consider the following event procedure written for the AnyObject_Select event.

```
Private Sub AnyObject_Select()
    Set MyObject = AnyObject
End Sub
```

If the variable MyObject is a global variable of type DiagramObject, then within the Select event you can set MyObject to the DiagramObject that is currently active. However, if you try to use MyObject after the event is over, it returns an error because an event is not in progress. Since you set MyObject to the AnyControl, your variable is pointing at the AnyControl that is dynamically pointing at the active object, which is nothing outside of an event.

If your intent is to hold on to the specific object that the AnyObject control is pointing at inside the event, then you need to use the PermanentDiagramObject property. This property gives you a DiagramObject that is valid after the event is over (outside of the event). The change to your code is as follows (MyDiagramObject is a global variable of type DiagramObject):

```
Private Sub AnyObject_Select()
    Set MyDiagramObject = AnyObject.PermanentDiagramObject
End Sub
```

Example The following example defines two subroutines and an event. The first subroutine "MakeShapes ( )" puts two shapes in the diagram. Go to the interface a select either one of the shapes. Selecting a shape triggers the AnyObject_BeforeSelect event, which then captures the DiagramObject for use outside the event. Next, run the second subroutine, which changes the fill color the permanent DiagramObject that was captured during the event.

```
Public igxDiagObj As DiagramObject

Public Sub MakeShapes()  ' Run first
    ' Add two shapes to the diagram
   ActiveDiagram.DiagramObjects.AddShape 1440, 1440
   ActiveDiagram.DiagramObjects.AddShape 1440 * 3, 1440
   ' Display a message that the event is ready
   MsgBox "Shapes created. The event is now active." _
        & Chr(13) & _
        "Return to the diagram and try selecting one of the shapes." _
        & Chr(13) & _
        "Then run the next subroutine."
End Sub

Public Sub ChangePermanentObject() 'Run second
    MsgBox "Click OK to change the permanent shape to white."
   igxDiagObj.Shape.FillColor = vbBlue
```

```
   MsgBox "Click OK to continue."
End Sub

Private Sub AnyObject_BeforeSelect(Cancel As Boolean)
    ' Point igxDiagObj at the same object as AnyObject
    Set igxDiagObj = AnyObject.PermanentDiagramObject
End Sub
```

See Also

PermanentConnectorLine property

PermanentDepartment property

PermanentDiagram property

PermanentDocument property

PermanentShape property

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

PropertyChange Event

Syntax **Private Sub** *DiagramObject*_**PropertyChange**(*Property* As Property)

Description The PropertyChange event occurs when the value of a property in a DiagramObject's property list changes. The event fires for every property in the property list that changes.The *Property* parameter contains the Property object that changed.

You can listen to the PropertyChange event to monitor when a property changes. You might use this event to break your code into modules—a form could modify a diagram property, and an event handler for the property change event in another module could respond to that change in some way.

You can create as many property lists as you want for a DiagramObject. Each property list can contain as many properties as you want. Each property has a name and a value.

When you name properties and property lists, be sure to pick names that are likely to be unique. One strategy is to use your company name followed by the property name or property list name. Also, be sure to check the name of the Parent of the property (the property list that the property is in). It is possible that there is another property list in the diagram that you are not aware of that, although named differently than your property list, uses some of the same names for individual properties.

Example The following example monitors property changes for a shape. If a property changes, it displays the property that was changed, and what it was changed to.

```
' Create a variable that hears DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension the variables
    Dim igxDiagram As Diagram
    Dim igxShape As Shape
    Dim igxProperty As Property
    Dim igxPropertyList As PropertyList
    Dim igxPropertyLists As PropertyLists
    ' Set igxDiagram variable to the Diagram object
    Set igxDiagram = Application.ActiveDiagram
    ' Set igxShape variable to a new Shape object
    Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
    ' Get the shape's diagram object
    Set igxDiagramObject = igxShape.DiagramObject
    ' Set the variable to the PropertyLists collection object
    Set igxPropertyLists = igxDiagramObject.PropertyLists
    ' Set the igxPropertyList variable to the PropertyList object
    Set igxPropertyList = igxPropertyLists.Add("Test List")
    ' Set the igxProperty variable to the Property object created
    Set igxProperty = igxPropertyList.Add("Test Property")
    ' Fill the property value
    igxProperty.Value = "XYZ"
End Sub

Private Sub igxDiagramObject_PropertyChange(ByVal Property As Property)
   MsgBox "The " & Property.Name & " property was changed to: " _
       & Property.Value
End Sub
```

See Also

Property object

PropertyList object

PropertyLists object

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

PropertyLists Property

Syntax *DiagramObject*.**PropertyLists**

Data Type PropertyLists collection object (read-only, See Object Properties)

Description The PropertyLists property returns the PropertyLists collection associated with the specified DiagramObject object. The PropertyLists collection provides access to any PropertyList objects and Property objects that are used by the DiagramObject.

Example The following example adds a property to a shape and displays the property's contents.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxDiagram As Diagram
   Dim igxShape As Shape
   Dim igxProperty As Property
   Dim igxPropertyList As PropertyList
   Dim igxPropertyLists As PropertyLists
   ' Set igxDiagram variable to the Diagram object
   Set igxDiagram = Application.ActiveDiagram
   ' Set igxShape variable to a new Shape object
   Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Get the shape's diagram object
   Set igxDiagramObject = igxShape.DiagramObject
   ' Set the variable to the PropertyLists collection object
   Set igxPropertyLists = igxDiagramObject.PropertyLists
   ' Set the igxPropertyList variable to the PropertyList object
   Set igxPropertyList = igxPropertyLists.Add("Test List")
   ' Set the igxProperty variable to the Property object created
   Set igxProperty = igxPropertyList.Add("Test Property")
   ' Fill the property value
   igxProperty.Value = "XYZ"
   MsgBox "The " & igxProperty.Name & " property is set to " _
       & igxProperty.Value
End Sub
```

See Also PropertyList object

PropertyLists object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Redraw Method

Syntax *DiagramObject*.**Redraw**([*UpdateNow* As Boolean = False])

Description The Redraw method forces the specified DiagramObject to repaint. The main purpose of this method is to assure that the DiagramObject is redrawn correctly after making changes to its properties, connection points, etc.

The *UpdateNow* argument is optional, and allows you to force the redraw to occur immediately.

Example The following example iterates through all the DiagramObjects in the diagram and repaints each one.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxDiagramObject As DiagramObject
' Set the Diagram variables to the new Diagram objects
Set igxDiagram = Application.ActiveDiagram
' Add a shapes to the diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Add another shape, but on the next page to the right
Set igxShape2 = igxDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
' Connect the shapes with a connector line
Set igxConnector = igxDiagram.DiagramObjects.AddConnectorLine(, , igxShape1,
ixDirEast, , , , igxShape2, ixDirWest)
' Redraw all the objects
MsgBox "Click OK to redraw the objects."
For Each igxDiagramObject In ActiveDiagram.DiagramObjects
    igxDiagramObject.Redraw
Next igxDiagramObject
MsgBox "Click OK to continue."
```

Resize Method

Syntax *DiagramObject*.**Resize** *Width* As Long, *Height* As Long

Description The Resize method allows you to resize a DiagramObject object. When resizing, the object maintains its current position relative to the top, left corner (based on the Top and Left properties).

Example The following example creates two shapes in the active diagram at the same location. The second shape's fill type is set to None so the two shapes can be seen together. The second shape is then resized to 2 x 3 inches.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape at the same position
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "View diagram"
' Set the fill type of Shape 2 to None so it can be seen
' through Shape 1
igxShape2.FillType = ixFillNone
MsgBox "View diagram"
' Resize the second shape to make it 2 inches by 3 inches
ActiveDiagram.DiagramObjects.Item(2).Resize 1440 * 2, 1440 * 3
MsgBox " Shape 2 resized to be 2 x 3 inches"
```

See Also

Bottom property

CenterX property

CenterY property

Left property

Right property

Top property

Move method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Right Property

Syntax *DiagramObject*.**Right**

Data Type Long (read/write)

Description The Right property specifies the location of the right side of the referenced DiagramObject object. The Right property and the Left property are mutually exclusive. The value of the one set most recently is applied. Values for this property are specified in twips (1440 twips = 1 inch).

Example The following example creates two shapes in the active diagram at the same location. The second shape's fill type is set to None so the two shapes can be seen together. The second shape is then resized to 2 x 3 inches. Finally, Shape2's Left property is set to 2 inches and then its Right property is set to 5 inches to move the shape.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape at the same position
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "View diagram"
' Set the fill type of Shape 2 to None
igxShape2.FillType = ixFillNone
MsgBox "View diagram"
' Resize the second shape to make it 2 inches by 3 inches
ActiveDiagram.DiagramObjects.Item(2).Resize 1440 * 2, 1440 * 3
' Sequentially set the Left, then the Right properties
' to move the shape
ActiveDiagram.DiagramObjects.Item(2).Left = 1440 * 2
MsgBox "Left side of shape at 2 inch mark"
ActiveDiagram.DiagramObjects.Item(2).Right = 1440 * 5
MsgBox "Right side of shape is at 5 inch mark"
```

See Also

Bottom property

CenterX property

CenterY property

Left property

Top property

Move method

Resize method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Select Event

Syntax **Private Sub** *DiagramObject*_**Select**()

Description The Select event occurs after the specified DiagramObject object has been selected.

Example The following example causes a shape to change color while it's selected.

```
' Dimension a module variable that hears DiagramObject events
Private WithEvents igxDiagramObject As DiagramObject

Private Sub Main()
    ' Dimension variables
    Dim igxShape1 As Shape
    ' Create a shape
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    ' Set the diagram object variable
    Set igxDiagramObject = igxShape1.DiagramObject
    MsgBox "The event is ready. Try selecting and deselecting " _
       "the shape."
End Sub

Private Sub igxDiagramObject_Deselect()
    igxDiagramObject.Shape.FillColor = vbWhite
End Sub

Private Sub igxDiagramObject_Select()
    igxDiagramObject.Shape.FillColor = vbYellow
End Sub
```

The following is another example that creates VBA controls for two shapes, and has a separate Select event for each shape..

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxDiagramObject As DiagramObject
Dim igxConnLine1 As ConnectorLine
Dim igxGraphic As TextGraphicObject
Dim igxTextObj As TextGraphicObject
Dim igxGraphicBuilder As New GraphicBuilder
Dim iCount As Integer
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right of Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "View the state of the diagram"
' Draw a direct connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
   (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
   ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
   ixDirWest, ixConnectRelativeToShape)
```

```
MsgBox "View the state of the diagram"
' Create a graphic consisting of a rectangle and an ellipse
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Add the graphic in the active diagram
Set igxGraphic = ActiveDiagram.DiagramObjects. _
   AddGraphic(igxGraphicBuilder.Graphic, 1440, 2880, 1440, 1440)
MsgBox "View the state of the diagram"
' Create a text object in the active diagram
Set igxTextObj = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440 * 4, Width:=1440, _
   Text:="I am a Text Object")
MsgBox "View the state of the diagram"
' Create VBA controls for the two shapes
For Each igxDiagramObject In ActiveDiagram.DiagramObjects
   If (igxDiagramObject.Type = ixObjectShape) Then
       MsgBox "Shape object found. Create VBA control."
       If (igxDiagramObject.IsVbaControl = False) Then
           igxDiagramObject.CreateVbaControl
       End If
   End If
Next igxDiagramObject
' Select each diagram object in the active diagram
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
   ActiveDiagram.DiagramObjects.Item(iCount).Selected = True
   If (iCount > 1) Then
       ActiveDiagram.DiagramObjects.Item(iCount - 1). _
           Selected = False
   End If
   MsgBox "View the diagram"
Next iCount
End Sub

Private Sub Shape1_Select()
    Shape1.FillType = ixFillSolid
    Shape1.FillColor = vbYellow
    Shape1.Text = "I have been selected"
    MsgBox "View the diagram"
End Sub

Private Sub Shape2_Select()
    Shape1.FillType = ixFillSolid
    Shape1.FillColor = vbYellow
    Shape1.Text = "I, too, have been selected"
    MsgBox "View the diagram"
End Sub
```

See Also Selected property

BeforeSelect event

Deselect event

Diagram.Selection property

Diagram.SelectionChange event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Selected Property

Syntax *DiagramObject*.**Selected**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Selected property, based on its value, causes the specified DiagramObject to be selected (True) or unselected (False). For instance, if you want to locate and select a particular shape on a diagram, you could search the DiagramObjects collection until the correct object was found, then set its Selected property to True. This allows you to perform actions on the object that require it to be selected, to draw the user's attention to the object for some purpose, or to fire one of the "selection" events.

Example The following example creates five diagram objects in the active diagram: 2 shapes, a connector, and 2 text-graphic objects. A For loop is then used to select each object in sequence.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxGraphic As TextGraphicObject
Dim igxTextObj As TextGraphicObject
Dim igxGraphicBuilder As New GraphicBuilder
Dim iCount As Integer
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right of Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(3))
MsgBox "View the state of the diagram"
' Draw a direct connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
   (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
   ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
   ixDirWest, ixConnectRelativeToShape)
MsgBox "View the state of the diagram"
' Create a graphic consisting of a rectangle and an ellipse
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbGreen
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Add the graphic in the active diagram
Set igxGraphic = ActiveDiagram.DiagramObjects. _
   AddGraphic(igxGraphicBuilder.Graphic, 1440, 2880, 1440, 1440)
MsgBox "View the state of the diagram"
' Create a text object in the active diagram
Set igxTextObj = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440 * 4, Width:=1440, _
   Text:="I am a Text Object")
MsgBox "View the state of the diagram"
' Select each diagram object in the active diagram
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
    ActiveDiagram.DiagramObjects.Item(iCount).Selected = True
   If (iCount > 1) Then
```

```
            ActiveDiagram.DiagramObjects.Item(iCount - 1). _
                Selected = False
        End If
        MsgBox "View the state of the diagram"
    Next iCount
```

See Also

BeforeSelect event

Select event

Deselect event

Diagram.Selection property

Diagram.SelectionChange event

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Shape Property

Syntax *DiagramObject*.**Shape**

Data Type Shape object (read-only, See Object Properties)

Description The Shape property returns a Shape object if the specified DiagramObject is a shape (Type property = ixObjectShape). When the DiagramObject is a shape, this property provides access to the properties, methods, and events for controlling a Shape object.

Use the Type property to determine if a DiagramObject is a Shape object. If the DiagramObject is not a Shape object, then this property returns the 'Nothing' value.

Example The following example creates five diagram objects in the active diagram: 2 shapes, a connector, and 2 text-graphic objects. A For Each loop is used to test the Type property of each diagram object to find all objects that are shapes. When a shape is found, the fill is changed to a solid green, and the ShapeClass and the Class ID are displayed in a message box.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxGraphic As TextGraphicObject
Dim igxTextObj As TextGraphicObject
Dim igxGraphicBuilder As New GraphicBuilder
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right of Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "View the state of the diagram"
' Draw a direct connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
   (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
   ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
   ixDirWest, ixConnectRelativeToShape)
MsgBox "View the state of the diagram"
' Create a graphic consisting of a rectangle and an ellipse
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Add the graphic in the active diagram
Set igxGraphic = ActiveDiagram.DiagramObjects. _
    AddGraphic(igxGraphicBuilder.Graphic, 1440, 2880, 1440, 1440)
MsgBox "View the state of the diagram"
' Create a text object in the active diagram
Set igxTextObj = ActiveDiagram.DiagramObjects. _
    AddTextObject(1440, 1440 * 4, Width:=1440, _
   Text:="I am a Text Object")
MsgBox "View the state of the diagram"
' Find any DiagramObjects that are of type TextGraphic
For Each DiagramObject In DiagramObjects
    If (DiagramObject.Type = ixObjectShape) Then
```

```
            MsgBox "Shape object found"
            ' Put a green solid fill in the shape and display
            ' its shape class and class ID
            DiagramObject.Shape.FillType = ixFillSolid
            DiagramObject.Shape.FillColor = vbGreen
            MsgBox "This shape is in ShapeClass " _
                & DiagramObject.Shape.ShapeClass.Name _
                & Chr$(13) & " and its Class ID is " _
                & DiagramObject.Shape.ShapeClass.ClassID
        End If
    Next DiagramObject
    MsgBox "View the state of the diagram"
```

See Also

Type property

Shape object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

TextGraphicObject Property

Syntax *DiagramObject*.**TextGraphicObject**

Data Type TextGraphicObject object (read-only, See Object Properties)

Description The TextGraphicObject property returns a TextGraphicObject object if the specified DiagramObject is a TextGraphic (Type property = ixObjectTextGraphic). When the DiagramObject is a TextGraphic, this property provides access to the properties, methods, and events for controlling a TextGraphicObject object.

Use the Type property to determine if a DiagramObject is a TextGraphicObject. If the DiagramObject is not a TextGraphicObject, then this property returns the 'Nothing' value.

Example The following example creates five diagram objects in the active diagram: 2 shapes, a connector, and 2 text-graphic objects. A For Each loop is used to test the Type property of each diagram object to find all objects that are TextGraphicObject objects. Then each TextGraphicObject is tested to determine whether it contains any text in its Text property. If so, the text string is changed.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxGraphic As TextGraphicObject
Dim igxTextObj As TextGraphicObject
Dim igxGraphicBuilder As New GraphicBuilder
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right of Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "View the state of the diagram"
' Draw a direct connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
MsgBox "View the state of the diagram"
' Create a graphic consisting of a rectangle and an ellipse
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
   FillFormat.FillColor = vbBlue
' Add the graphic in the active diagram
Set igxGraphic = ActiveDiagram.DiagramObjects. _
   AddGraphic(igxGraphicBuilder.Graphic, 1440, 2880, 1440, 1440)
MsgBox "View the state of the diagram"
' Create a text object in the active diagram
Set igxTextObj = ActiveDiagram.DiagramObjects. _
    AddTextObject(1440, 1440 * 4, Width:=1440, _
    Text:="I am a Text Object")
MsgBox "View the state of the diagram"
' Find any DiagramObjects that are of type TextGraphic
For Each DiagramObject In DiagramObjects
```

```
    If (DiagramObject.Type = ixObjectTextGraphic) Then
        MsgBox "TextGraphic object found"
        ' If the object contains text, change the text
        If (DiagramObject.TextGraphicObject.Text <> "") Then
            MsgBox "Found the Text-only object"
            DiagramObject.TextGraphicObject.Text = _
                "I have been changed"
            MsgBox "View the state of the diagram"
        End If
    End If
Next DiagramObject
```

See Also

Type property

TextGraphicObject object

iGrafx API Object Hierarchy

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

Type Property

Syntax *DiagramObject*.**Type**

Data Type IxObjectType enumerated constant (read-only)

Description The Type property indicates the “type” of a DiagramObject object; that is, is it a shape, a connector line, a department, etc. This property provides a way, when accessing the DiagramObjects collection, of determining the object type.

The IxObjectType constant defines valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixObjectShape
1	ixObjectDepartment
3	ixObjectOle
4	ixObjectConnector
5	ixObjectTextGraphic
6	ixObjectGroup
7	ixObjectLegend
8	ixObjectOther

Example The following example adds several objects to a diagram. Information about each DiagramObject is gathered into a string, and the results are displayed.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnector As ConnectorLine
    Dim igxLegend As Legend
    Dim igxText As TextGraphicObject
    Dim Object As DiagramObject
    ' Add several objects to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 4, 1440)
    Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, _
       ixDirWest)
    Set igxLegend = ActiveDiagram.DiagramObjects.AddLegend _
       (1440 * 2, 1440 * 4)
    Set igxText = ActiveDiagram.DiagramObjects.AddTextObject _
       (1440, 1440 * 5, , , "Text Graphic")
    ' Collect info on each object into a string
    For Each Object In ActiveDiagram.DiagramObjects
        sString = sString & "Object ID# " & Object.ID & " - " _
            & GetType(Object) & Chr(13)
    Next Object
    ' Display the result
    MsgBox "The Diagram contains these objects: " & Chr(13) _
       & Chr(13) & sString
```

```
End Sub

Private Function GetType(Object As DiagramObject) As String
    ' Determine the object's type, and return a string
    Select Case Object.Type
        Case ixObjectConnector
            GetType = "Connector Line"
        Case ixObjectLegend
            GetType = "Legend"
        Case ixObjectShape
            GetType = "Shape"
        Case ixObjectTextGraphic
            GetType = "Text Graphic"
    End Select
End Function
```

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

UpdateFields Method

Topic Under Construction!!!

Syntax *DiagramObject*.**UpdateFields**

Description The UpdateFields method

Example The following example

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

UserEvent Event

Syntax **Private Sub** *DiagramObject*_**UserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The UserEvent event provides a means of implementing your own custom events. Your custom events can then be triggered with the FireUserEvent method, which fires the specified "UserEvent" on the document. You can use this functionality to send messages to any objects listening to document-level events.

You must pick an event identifier string to use for your event. You might choose to use something like your company name followed by the event name. You should choose a name that won't conflict with names picked by other developers.

You can pass one parameter to the event. This parameter is a Variant, so one logical choice is to pass a class.

You then write code in a UserEvent handler to perform some actions when your event fires. This code should be of the form:

```
If EventIdentifier = "<<Your identifier string>>" Then
   << Write your code here >>
End If
```

Example The following example defines a new user event called "ShowUsers". The *Parameter* that gets passed is a class, which has one property called Count. The event handler displays the passed parameter's Count property.

The following code creates a simple class with one property. Create a new class within a diagram project called Class1, and copy this code into it.

```
' Class1
' It contains one property, read only
Public Property Get Count() As Long
   Count = 25
End Property
```

The following code is the main program. Copy this, and the UserEvent subroutine, into the diagram project code window

```
' Run this subroutine to test the event
Public Sub Main()
   ' Fire the UserEvent
   Application.FireUserEvent "ShowUsers", New Class1
End Sub

' This event handler runs every time any FireUserEvent method
' is used in the system
Private Sub Application_UserEvent(ByVal EventIdentifier As String, ByVal
Parameter As Variant)
    ' Check if the Identifier string is the one we want
    If EventIdentifier = "ShowUsers" Then
       ' Redirect to Class1
       MsgBox "The number of users is " & Parameter.Count
    End If
End Sub
```

See Also FireUserEvent method

{button DiagramObject object,JI(`igrafxrf.HLP',`DiagramObject_Object')}

DiagramObjects Object

The DiagramObjects object is a collection of individual DiagramObject objects. Each Diagram has a DiagramObjects collection associated within it. Its purpose is to provide access to individual DiagramObjects that are in a particular Diagram. It is through this object that a DiagramObject is added to the diagram.

The DiagramObjects object provides the following functionality:

- The ability to access any DiagramObject objects that have created in a Diagram.
- The ability to determine how many DiagramObject objects are in the collection.
- The ability to add a new DiagramObject object to a Diagram (new shapes, connector lines, graphics, etc.).

The following example gets the DiagramObjects collection from the ActiveDiagram object.

```
' Dimension the variables
Dim igxDiagramObjects As DiagramObjects
' Get the DiagramObjects collection
Set igxDiagramObjects = Application.ActiveDiagram.DiagramObjects
```

For more information about what DiagramObject objects are, and how and why they are used, refer to the DiagramObject object.

Properties, Methods, and Events

All of the properties, methods, and events for the DiagramObjects object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	AddActiveXControl	
Count	AddConnectorLine	
ObjectRange	AddGraphic	
Parent	AddLegend	
	AddOleObject	
	AddShape	
	AddTextObject	
	Item	

AddActiveXControl Method

Topic Under Construction!!!

Syntax *DiagramObjects*.**AddActiveXControl**(*ProgID* As String, *CenterX* As Long, *CenterY* As Long) As OleObject

Description The AddActiveXControl method

See Also OleObject object

{button DiagramObjects object,JI(`igrafxrf.HLP',`DiagramObjects_Object')}

AddConnectorLine Method

Topic Under Construction!!! -- Example should work, but does not

Syntax

DiagramObjects.**AddConnectorLine**([*RouteType* As IxRouteType], [*RouteFlag* As IxRouteFlag = ixRouteFlagFindEdge], [*SourceShape* As IXShape], [*SourceDir* As IxDirection], [*SourceConnectType* As IxConnectType = ixConnectRelativeToShape], [*SourceX* As Long = -1], [*SourceY* As Long = -1], [*DestShape* As IXShape], [*DestDir* As IxDirection], [*DestConnectType* As IxConnectType = ixConnectRelativeToShape], [*DestX* As Long = -1], [*DestY* As Long = -1]) As Connector

Description

The AddConnectorLine method creates and adds a ConnectorLine object to the DiagramObjects collection for a particular diagram. The method returns a ConnectorLine object that can be stored in a variable or ignored.

Connector lines can be drawn many different ways, such as between two shapes, between two designated points on a diagram, or between any combination of point and object. The arguments for this method are described below. Use whichever combination of these arguments are necessary to create the appropriate connector line.

The *RouteType* argument specifies an IxRouteType constant that indicates the type of routing to use for the connector line. Valid values for this argument are listed in the following table (a * indicates the default).

Value	Name of Constant	Description
0	ixRouteDirect	Draws a direct, straight line.
1	ixRouteRightAngle *	Draws a routed line where each change of direction is at a right angle.
2	ixRouteCurved	Draws a curved line where each turn is a curve.
3	ixRouteOrgChart	Draws a line in the style of an organizational chart.
4	ixRouteCauseAndEffect	Draws a Ishikawa cause and effect line. Cause and Effect lines do not connect to shapes.
5	ixRouteLightningBolt	Draws a line in the style of a lightning bolt.

The *RouteFlag* argument specifies whether to extend the connector to the edge of a graphic when there are no connect points physically on the graphic's edge (that is, the connect points are on the bounding box or some other location). This argument has no effect unless the connector line is being routed to a shape. The IxRouteFlag constant defines the valid values for this argument, and are listed in the following table (a * indicates the default). For more information about this argument, refer to the ConnectorLine object.

This argument is most useful for shapes with curved or slanted edges where the edge of the shape does not coincide with the edge of the bounding rectangle.

Value	Name of Constant	Description
0	ixRouteFlagDontFindEdge	Prevents iGrafx Professional from calculating the extension of the connector line to the actual edge of a shape. The connector line is routed to a point on the shape's bounding box.
1	ixRouteFlagFindEdge *	Causes iGrafx Professional to

	calculate the extension of the connector line so that it actually attaches to the shape's boundary edge rather than to the edge of the shape's bounding box.

The *SourceShape* and *DestShape* arguments specify the Shape object that is to be the Source or Destination of the connector line, respectively. Both of these arguments are optional.

The *SourceDir* argument specifies the direction from which the connector line exits the source shape (based on a compass heading system; North is always up—to the top of the screen). The *DestDir* argument specifies the direction the connector line enters the destination shape. Both of these arguments are optional, and have no effect unless the connector line is attached to a shape. The IxDirection constant defines the valid values for both the *SourceDir* and *DestDir* arguments, and are listed in the following table.

Value	Name of Constant
1	ixDirNorth
2	ixDirEast
3	ixDirSouth
4	ixDirWest

The *SourceX* and *SourceY* arguments specify the horizontal and vertical positions, respectively, of the starting point for the connector line. The units of measure are twips (1440 twips = 1 inch). This argument is not used if the *SourceShape* argument is specified.

The *DestX* and *DestY* arguments specify the horizontal and vertical positions, respectively, of the ending point for the connector line. The units of measure are twips (1440 twips = 1 inch). This argument is not used if the *DestShape* argument is specified.

The *SourceConnectType* and *DestConnectType* arguments specify how the connection to the shape is maintained when the shape is resized. This argument has no effect unless the connector line is attached to a shape. The IxConnectType constant defines the valid values for this argument, and are listed in the following table (a * indicates the default).

Value	Name of Constant	Description
0	ixConnectRelativeToShape *	Connects to a relative point on the shape – meaning that if it is connected at the middle of the left edge, it stays connected at the middle of the left edge when the shape is resized.
1	ixConnectAbsoluteFromTopLeft	Connects to a point a fixed distance from the top left – if it connects 0.5 inches from the top on the left edge, when resized, it stays 0.5 inches from the top on the left edge.

Example

The following example creates three shapes and two connector lines. It shows the effect of intersecting connector lines, and shows various route types, such as RightAngle, Direct, and Curved.

```
ConnectType and RouteFlag methods are not working. Also, it is failing to show
```

```
the Intersection style

' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxDiagObj As DiagramObject
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxBuilder As New GraphicBuilder
Dim lShapeTop As Long
' Add several objects to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 3)
igxShape1.Text = "1"
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape2.Text = "2"
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440, _
    Application.ShapeLibraries.Item(1).Item(4))
igxShape3.Text = "3"
' Add a connector line
Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape2, _
    ixDirEast)
' Show all of the routing types
MsgBox "RightAngle routing"
igxConnector1.Routing = ixRouteDirect
MsgBox "Direct routing"
igxConnector1.Routing = ixRouteCurved
MsgBox "Curved routing"
igxConnector1.Routing = ixRouteCauseAndEffect
MsgBox "Cause and Effect routing"
igxConnector1.Routing = ixRouteOrgChart
MsgBox "OrgChart routing"
igxConnector1.Routing = ixRouteLightningBolt
MsgBox "Lightning Bolt routing"
igxConnector1.Routing = ixRouteRightAngle
MsgBox "Return the routing to RightAngle"
' Set the intersection style and color
ActiveDiagram.IntersectionStyle = ixIntersectionLargeSquare
ActiveDiagram.IntersectionColor = vbRed
' Add a second connector line that intersects the first
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape3, _
    ixDirWest)
MsgBox "Intersecting Connector Lines. Now remove both lines."
' Remove the connector lines
For Each igxDiagObj In ActiveDiagram.DiagramObjects
    If (igxDiagObj.Type = ixObjectConnector) Then
        igxDiagObj.DeleteDiagramObject
    End If
Next igxDiagObj
MsgBox "Lines removed"
' Show routing that finds the edge of a shape and then so it
' does not find the edge of the shape. First, replace the graphic
```

```
' in Shape 3
' Create a polygon by drawing lines
igxBuilder.BeginPath
igxBuilder.MoveTo 0.1, 1
igxBuilder.LineTo 0.3, 0
igxBuilder.LineTo 0.9, 0
igxBuilder.LineTo 0.7, 1
igxBuilder.Close
igxBuilder.EndPath
' Replace the graphic
igxShape3.Graphic.Replace igxBuilder.Graphic
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirNorth, , , , igxShape3, ixDirWest)
MsgBox "Connector routed so it finds the edge of the shape"
' Remove the connector line
For Each igxDiagObj In ActiveDiagram.DiagramObjects
    If (igxDiagObj.Type = ixObjectConnector) Then
        igxDiagObj.DeleteDiagramObject
    End If
Next igxDiagObj
MsgBox "Line removed"
' Route so don't find the edge
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagDontFindEdge, igxShape1, _
    ixDirNorth, , , , igxShape3, ixDirWest)
MsgBox "Connector routed so it does not find the edge of the shape"
' Remove the connector line
For Each igxDiagObj In ActiveDiagram.DiagramObjects
    If (igxDiagObj.Type = ixObjectConnector) Then
        igxDiagObj.DeleteDiagramObject
    End If
Next igxDiagObj
MsgBox "Line removed"
' Route a line from shape 1 to a point in the diagram
' This uses the DestX and DestY arguments
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagDontFindEdge, igxShape1, ixDirNorth, _
    DestX:=1440 * 5, DestY:=1440 * 3)
MsgBox "Connector routed to a point in the diagram"
' Remove the connector line
For Each igxDiagObj In ActiveDiagram.DiagramObjects
    If (igxDiagObj.Type = ixObjectConnector) Then
        igxDiagObj.DeleteDiagramObject
    End If
Next igxDiagObj
MsgBox "Line removed"
' Connect to Shape 3 so that the Connect Type is Relative to the Shape
' then switch it so it is Absolute from top left
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, ixDirNorth, _
    ixConnectRelativeToShape, , , igxShape3, ixDirWest)
MsgBox "Connected Relative to shape. Watch the position of the connection" _
    & Chr(13) & "when the shape is resized."
lShapeTop = igxShape3.DiagramObject.Top
```

```
igxShape3.DiagramObject.Height = 3000
MsgBox "View the connector line location." & Chr(13) & Chr(13) _
    & "Reset the shape's size and remove the connector."
igxShape3.DiagramObject.Top = lShapeTop
igxShape3.DiagramObject.Height = 1080
' Remove the connector line
For Each igxDiagObj In ActiveDiagram.DiagramObjects
    If (igxDiagObj.Type = ixObjectConnector) Then
        igxDiagObj.DeleteDiagramObject
    End If
Next igxDiagObj
MsgBox "Line removed"
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, ixDirNorth, _
    ixConnectAbsoluteFromTopLeft, , , igxShape3, ixDirWest)
MsgBox "Connected Relative to shape. Watch the position of the connection" _
    & Chr(13) & "when the shape is resized."
lShapeTop = igxShape3.DiagramObject.Top
igxShape3.DiagramObject.Height = 3000
igxShape3.DiagramObject.Top = lShapeTop
MsgBox "End of Example"
```

See Also ConnectorLine object

{button DiagramObjects object,JI(`igrafxrf.HLP',`DiagramObjects_Object')}

AddGraphic Method

Syntax *DiagramObjects*.**AddGraphic**(*Graphic* As Graphic, *CenterX* As Long, *CenterY* As Long, *Width* As Long, *Height* As Long, [*SnapToGrid* As Boolean = False]) As TextGraphicObject

Description The AddGraphic method adds a TextGraphicObject object to the DiagramObjects collection of a diagram. The referenced diagram does not have to be the active diagram. The result of the AddGraphic method can be assigned to a TextGraphicObject variable or can be ignored. The arguments for this method are described below.

Even though this method returns a TextGraphicObject object, its primary purpose is to add a graphic to a diagram. To add text to a diagram as a TextGraphicObject object, use the AddTextObject method.

The *Graphic* argument is a Graphic object that represents the graphic to be added to the DiagramObjects collection. This graphic is normally created using the GraphicsBuilder object, or it can be grabbed from another object with a graphic.

The *CenterX* and *CenterY* arguments specify where to place the center of the TextGraphic object on the diagram. The units of measure are twips (1440 twips = 1 inch).

The *Width* and *Height* arguments specify the size of the bounding rectangle of the TextGraphic object. The units of measure are twips (1440 twips = 1 inch).

The *SnapToGrid* argument specifies whether to snap the object to the grid. Setting this argument to True potentially can change the CenterX and CenterY position you have chosen so that it snaps to the grid.

Example The following example adds a graphic created using the GraphicBuilder object to the DiagramObjects collection.

```
' Dimension the variables
Dim igxGraphic As TextGraphicObject
Dim igxGraphicBuilder As New GraphicBuilder
' Create a graphic consisting of a rectangle, an ellipse,
' and a star, and their fill colors to red, blue, and green
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
   FillFormat.FillColor = vbBlue
igxGraphicBuilder.Star 0.5, 0.5, 0.3, 0.15, 5, 30
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(3). _
   FillFormat.FillColor = vbGreen
' Create a graphic in the active diagram
Set igxGraphic = ActiveDiagram.DiagramObjects. _
   AddGraphic(igxGraphicBuilder.Graphic, 1440, 1440, 1440, 1440)
MsgBox "Added a TextGraphicObject to the diagram"
```

See Also AddTextObject method

Graphic object

GraphicBuilder object

{button DiagramObjects object,JI(`igrafxrf.HLP',`DiagramObjects_Object')}

AddLegend Method

Syntax *DiagramObjects*.**AddLegend**(*Left* As Long, *Top* As Long) As Legend

Description The AddLegend method adds a Legend object to the DiagramObjects collection of a diagram. A Legend object is used to display CustomDataDefinitions for the document. The method returns a Legend object.

The referenced diagram does not have to be the active diagram. The result of the AddLegend method can be assigned to a Legend object variable, or can be ignored. The arguments for this method are described below.

The *Left* and *Top* arguments specify where to place the Legend object on the diagram. The units of measure are twips (1440 twips = 1 inch).

Example The following example creates a new Legend object in the active diagram, placed with the left edge at 2 inches and the top edge at 1 inch.

```
' Dimension the variables
Dim igxLegend As Legend
' Create a Legend object in the diagram
Set igxLegend = ActiveDiagram.DiagramObjects.AddLegend _
    (1440 * 2, 1440)
igxLegend.FillFormat.FillColor = vbRed
MsgBox "Added a Legend object to the diagram"
```

See Also Legend object

{button DiagramObjects object,JI(`igrafxrf.HLP',`DiagramObjects_Object')}

AddOleObject Method

Syntax *DiagramObjects*.**AddOleObject**(*FileName* As String, *CenterX* As Long, *CenterY* As Long, [*AsIcon* As Boolean = False], [*AsLink* As Boolean = False]) As OleObject

Description The AddOleObject method adds an OleObject to the DiagramObjects collection of a diagram. The AddOleObject method returns an OleObject that can be assigned to an OleObject variable or ignored.

The OLE object is drawn on the referenced diagram, which does not have to be an active diagram. The result of the AddOleObject method must be assigned to an OleObject variable. The arguments for this method are described below.

The *FileName* argument is the name of the file to be added as an OleObject. If an invalid name is supplied, then a run-time error occurs.

The *CenterX* and *CenterY* arguments specify where to place the center of the OleObject on the diagram. The units of measure are twips (1440 twips = 1 inch).

The *AsIcon* argument is a Boolean value that specifies whether to add the OLE object as an icon. The argument is optional.

The *AsLink* argument is a Boolean value that specifies whether to use OLE-linking. If the value is False, the object is embedded instead of linked. The argument is optional.

Example The following example adds a Word Document to the active diagram as an OleObject. This example requires that the computer has a folder called C:\My Documents, and that the folder contains a Word document called Sample.doc.

```
' Dimension the variables
Dim igxOleObject As OleObject
' Add an OLE object to the diagram
Set igxOleObject = ActiveDiagram.DiagramObjects.AddOleObject _
    ("C:\My Documents\sample.doc", 1440 * 3, 1440 * 3, False, False)
' Display the OLE object's classname
MsgBox "A " & igxOleObject.ClassName & _
    " has been added to the diagram."
```

See Also OleObject object

{button DiagramObjects object,JI(`igrafxrf.HLP',`DiagramObjects_Object')}

AddShape Method

Syntax *DiagramObjects*.**AddShape** (*CenterX* As Long, *CenterY* As Long, [*Shape* As ShapeLibraryItem], [*SnapToGrid* As Boolean = False]) As Shape

Description The AddShape method adds a Shape object to the DiagramObjects collection of a diagram. The referenced diagram does not have to be the active diagram. The result of the AddShape method must be assigned to a Shape variable. The AddShape method returns a Shape that can be assigned to a variable or ignored. The arguments for this method are described below.

The *CenterX* and *CenterY* arguments specify where to place the center of the shape on the diagram. The units are twips (1440 twips = 1 inch).

The *Shape* argument specifies the type of shape to add. This can be any valid shape from a (loaded) Shape Library. If the *Shape* argument is not specified, the currently selected Shape Library shape is used.

The *SnapToGrid* argument specifies whether to snap the object to the grid. Setting this argument to True potentially can change the CenterX and CenterY position you have chosen so that the shape snaps to the grid.

Example The following example adds two shapes to the active diagram, and connects them with a direct connector line. Each shape is added from the first ShapeLibrary. Shape 1 is the first shape in that library, and shape 2 is the second from that library.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
MsgBox "Click OK to add shapes to the diagram."
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(2), True)
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
MsgBox "Click OK to continue."
```

See Also Shape object

ShapeLibrary object

ShapeLibraryItem object

{button DiagramObjects object,JI(`igrafxrf.HLP',`DiagramObjects_Object')}

AddTextObject Method

Syntax *DiagramObjects*.**AddTextObject** (*Left* As Long, *Top* As Long, [*Width* As Long = -1], [*Height* As Long = -1], [*Text* As String = "0"], [*SnapToGrid* As Boolean = False]) As TextGraphicObject

Description The AddTextObject method adds a TextGraphicObject object to the DiagramObjects collection of a diagram. The text created has the characteristics, font, attributes, and color, of the current text defaults for the diagram. The method returns a TextGraphicObject object, which must be assigned to a TextGraphicObject variable.

The *Left* and *Top* arguments specify where to place the TextGraphicObject on the diagram. The units are twips (1440 twips = 1 inch).

The *Width* and *Height* arguments specify the size of the bounding rectangle of the TextGraphicObject. The units of measure are twips (1440 twips = 1 inch). The *Width* and *Height* arguments are optional.

The *Text* argument specifies the text to be displayed in the TextGraphicObject. This argument is optional and defaults to an empty string.

The *SnapToGrid* argument specifies whether to snap the object to the grid. Setting the value to True potentially can change the CenterX and CenterY position you have chosen so that the TextGraphicObject snaps to the grid.

Example The following example creates a TextGraphicObject on the active diagram by adding it to the DiagramObjects collection.

```
' Dimension the variables
Dim igxTGObject As TextGraphicObject
' Create the text on the active diagram.
Set igxTGObject = ActiveDiagram.DiagramObjects. _
   AddTextObject(1440, 1440, Text:="I am a TextBlock")
```

See Also TextGraphicObject object

{button DiagramObjects object,JI(`igrafxrf.HLP',`DiagramObjects_Object')}

Item Method

Syntax *DiagramObjects*.**Item**(*Index* As Integer) As DiagramObject

Description The Item method returns the DiagramObject object at the specified *Index* from the DiagramObjects collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type DiagramObject. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example uses the Item method and Count property to iterate through the objects in the diagram and output the ObjectName of each object.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim sString As String
' Add two shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 3, 1440)
' Assign object names to the shapes
igxShape1.DiagramObject.ObjectName = "Shape 1"
igxShape2.DiagramObject.ObjectName = "Shape 2"
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Assign an object name to the connector line
igxConnector.DiagramObject.ObjectName = "Connector Line"
' Collect the names of the DiagramObjects into a string
For Index = 1 To ActiveDiagram.DiagramObjects.Count
    sString = sString & ActiveDiagram.DiagramObjects _
       .Item(Index).ObjectName & Chr(13)
Next Index
' Display the result
MsgBox "The diagram contains these objects:" & Chr(13) & Chr(13) & _
    sString
```

{button DiagramObjects object,JI(`igrafxrf.HLP',`DiagramObjects_Object')}

ObjectRange Property

Syntax *DiagramObjects*.**ObjectRange**

Data Type ObjectRange object (read-only, See Object Properties)

Description The ObjectRange property returns an ObjectRange object that contains all of the objects in the specified DiagramObjects collection. The object range allows all of the objects in the DiagramObjects collection to be acted on by the methods and properties of the ObjectRange object.

Example The following example uses the ObjectRange property of the DiagramObjects collection to select all of the objects on the active diagram. This is accomplished by using the Selection.AddRange method with the ObjectRange property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim sString As String
' Add two shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "The ObjectRange property of the DiagramObjects object " _
    & Chr(13) & "contains " & _
    ActiveDiagram.DiagramObjects.ObjectRange.Count _
    & " objects."
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440)
MsgBox "The ObjectRange property of the DiagramObjects object " _
    & Chr(13) & "contains " & _
    ActiveDiagram.DiagramObjects.ObjectRange.Count _
    & " objects."
' Assign object names to the shapes
igxShape1.DiagramObject.ObjectName = "Shape 1"
igxShape2.DiagramObject.ObjectName = "Shape 2"
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
MsgBox "The ObjectRange property of the DiagramObjects object " _
    & Chr(13) & "contains " & _
    ActiveDiagram.DiagramObjects.ObjectRange.Count _
    & " objects."
' Assign an object name to the connector line
igxConnector.DiagramObject.ObjectName = "Connector Line"
MsgBox "Click OK to select all the DiagramObject objects."
' Select all the DiagramObject objects using the ObjectRange property
ActiveDiagram.Selection.AddRange _
    ActiveDiagram.DiagramObjects.ObjectRange
' Display the result
MsgBox "Click OK to continue."
```

See Also ObjectRange object

iGrafx API Object Hierarchy

{button DiagramObjects object,JI(`igrafxrf.HLP',`DiagramObjects_Object')}

DiagramView Object

The DiagramView object controls the view of a diagram through a window. The diagram view is iGrafx Professional's built-in view. If you run iGrafx Professional interactively, and select a new Basic Diagram from the opening dialog, you get a window with a view into Document1 - Diagram 1, that has the view positioning set to Top and Left equal 0. This is the initial diagram view. You can add any number of additional views into the same diagram (see the Views.AddDiagramView method).

Using the DiagramView object you can:

- Position the view to reveal any part of the diagram
- Center the view on a particular object
- Scroll to a particular object or page
- Zoom in or out on the view

Each DiagramView is an independent view into a diagram. Each DiagramView is numbered with an index number, and the index number appears in the Title Bar of the Window (ex. "Document – DiagramName:2")

The width and height of the view window affect the Width and Height properties of a DiagramView object, but not visa versa. If you increase the width of the view window, the DiagramView.Width property increases. If you increase the DiagramView.Width property, the view window width does not increase. Instead, the view window remains the same size, but the ZoomPercentage of the DiagramView increases to accommodate the change.

The ZoomPercent, Width, Height, Left, Top, CenterX, and CenterY properties are closely related. The following table describes how each property effects other properties.

Changing this property:	**Automatically changes these properties:**
CenterX	Left
CenterY	Top
Height	ZoomPercentage, Top, Left
Left	CenterX
Top	CenterY
Width	ZoomPercentage, Top, Left
ZoomPercentage	Width, Height, Top, Left

The following example adds some DiagramObjects to the diagram, and creates a new DiagramView. It then changes the Width and Height properties of the DiagramView, to demonstrate their relationship to the ZoomPercentage property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxView1 As DiagramView
Dim igxView2 As DiagramView
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape1.DiagramObject.ObjectName = "Shape 1"
' Add a shape
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
igxShape2.DiagramObject.ObjectName = "Shape 2"
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
  (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
```

```
igxConnector.DiagramObject.ObjectName = "Connector Line"
' Get the current DiagramView
Set igxView1 = ActiveDiagram.Views.Item(1).DiagramView
' Add a new DiagramView
Set igxView2 = ActiveDiagram.Views.AddDiagramView(ActiveDiagram)
' Activate the new DiagramView
igxView2.View.Window.Activate
' Try changing the Width and Height properties, and view the results
MsgBox "Click OK to increase the Width property."
igxView2.Width = igxView2.Width + 2000
MsgBox "ZoomPercent decreased as a result." & Chr(13) & _
  "Now click OK to decease the Height property."
igxView2.Height = igxView2.Height - 2000
MsgBox "ZoomPercent increased as a result."
```

Properties, Methods, and Events

All of the properties, methods, and events for the DiagramView object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	PointToScreen	
CenterX	PrintPreview	
CenterY	ScrollToObject	
FitTo	ScrollToPage	
Height		
Left		
Parent		
Top		
View		
Width		
ZoomPercentage		

CenterX Property

Syntax *DiagramView*.**CenterX**

Data Type Long (read/write)

Description The CenterX property specifies the CenterX position of the view in twips (1440 twips = 1 inch).

Example The following example centers the view on a shape. It illustrates the CenterX and CenterY property of the DiagramView and the CenterX and CenterY property of a DiagramObject.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxView1 As DiagramView
Dim igxView2 As DiagramView
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape1.DiagramObject.ObjectName = "Shape 1"
' Add a shape
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
igxShape2.DiagramObject.ObjectName = "Shape 2"
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
igxConnector.DiagramObject.ObjectName = "Connector Line"
' Get the current DiagramView
Set igxView1 = ActiveDiagram.Views.Item(1).DiagramView
' Add a new DiagramView
Set igxView2 = ActiveDiagram.Views.AddDiagramView(ActiveDiagram)
' Activate the new DiagramView
igxView2.View.Window.Activate
' Change the properties and view the result
MsgBox "Click OK to center on Shape 2."
igxView2.ZoomPercentage = 200
igxView2.CenterX = igxShape2.DiagramObject.CenterX
igxView2.CenterY = igxShape2.DiagramObject.CenterY
MsgBox "Click OK to continue."
```

See Also CenterY property

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

CenterY Property

Syntax *DiagramView*.**CenterY**

Data Type Long (read/write)

Description The CenterY property specifies the CenterY position of the view in twips (1440 twips = 1 inch).

Example Refer to the example for the CenterX property.

See Also CenterX property

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

FitTo Property

Syntax *DiagramView*.**FitTo**

Data Type IxFitTo enumerated constant (read/write)

Description The FitTo property specifies a method for zooming the view. The FitTo property provides a convenient way to zoom the view based on the diagram page width, or to see the entire diagram. The FitTo property affects the ZoomPercentage property, calculating the zoom percentage in order to fit the item indicated by the FitTo property's value.

The IxFitTo constant defines valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixFitToNormal
1	ixFitToPageWidth
2	ixFitToAll

Example The following example sets the diagram view to each of the three FitTo options.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxView As DiagramView
' Add several objects to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 3)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 10)
' Add connector lines
Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape2, _
    ixDirEast)
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape3, _
    ixDirWest)
Set igxView = ActiveDiagram.Views.Item(1).DiagramView
igxView.ZoomPercentage = 150
igxView.FitTo = ixFitToNormal
MsgBox "ixFitToNormal"
igxView.FitTo = ixFitToPageWidth
MsgBox "ixFitToPageWidth"
igxView.FitTo = ixFitToAll
MsgBox "ixFitToAll"
```

See Also ZoomPercentage property

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

Left Property

Syntax *DiagramView*.**Left**

Data Type Long (read/write)

Description The Left property sets the Left position of the view in twips (1440 twips = 1 inch).

Example The following example increases the Left property, and shows the effect on the CenterX property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxView1 As DiagramView
Dim igxView2 As DiagramView
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape1.DiagramObject.ObjectName = "Shape 1"
' Add a shape
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
igxShape2.DiagramObject.ObjectName = "Shape 2"
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
igxConnector.DiagramObject.ObjectName = "Connector Line"
' Get the current DiagramView
Set igxView1 = ActiveDiagram.Views.Item(1).DiagramView
' Add a new DiagramView
Set igxView2 = ActiveDiagram.Views.AddDiagramView(ActiveDiagram)
' Activate the new DiagramView
igxView2.View.Window.Activate
' Change the property and view the result
MsgBox "CenterX value is " & igxView2.CenterX & Chr(13) & _
"Click OK to increase the Left property."
igxView2.Left = igxView2.Left + 2000
MsgBox "CenterX value is " & igxView2.CenterX
```

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

PointToScreen Method

Syntax *DiagramView*.**PointToScreen**(*X* As Long, *Y* As Long, *ScreenX* As Long, *ScreenY* As Long)

Description The PointToScreen method converts a point in twips to the position in pixels on the screen. You supply the *X* and *Y* arguments, in units of twips, as the positions in the diagram view to be converted to pixels. The *ScreenX* and *ScreenY* parameters return the screen position in pixels.

Example The following example gives you the left and top position of the current selection in pixels (screen coordinates).

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxView1 As DiagramView
Dim igxView2 As DiagramView
Dim ScreenX As Long
Dim ScreenY As Long
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape1.DiagramObject.ObjectName = "Shape 1"
' Add a shape
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
igxShape2.DiagramObject.ObjectName = "Shape 2"
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
igxConnector.DiagramObject.ObjectName = "Connector Line"
' Get the current DiagramView
Set igxView1 = ActiveDiagram.Views.Item(1).DiagramView
' Add a new DiagramView
Set igxView2 = ActiveDiagram.Views.AddDiagramView(ActiveDiagram)
' Activate the new DiagramView
igxView2.View.Window.Activate
' Select a shape
ActiveDiagram.Selection.Add igxShape1.DiagramObject
' Convert diagram position Twips to screen position pixels
igxView2.PointToScreen igxShape1.DiagramObject.Top, _
   igxShape1.DiagramObject.Left, ScreenX, ScreenY
' Display the result
MsgBox "The Left and Top positions are " & ScreenX & ", " & ScreenY
```

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

PrintPreview Method

Syntax *DiagramView*.**PrintPreview**

Description The PrintPreview method opens the print preview window for the specified diagram view.

Example The following example sets up a DiagramView, and then displays the Print Preview.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxView1 As DiagramView
Dim igxView2 As DiagramView
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape1.DiagramObject.ObjectName = "Shape 1"
' Add a shape
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
igxShape2.DiagramObject.ObjectName = "Shape 2"
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
igxConnector.DiagramObject.ObjectName = "Connector Line"
' Get the current DiagramView
Set igxView1 = ActiveDiagram.Views.Item(1).DiagramView
' Add a new DiagramView
Set igxView2 = ActiveDiagram.Views.AddDiagramView(ActiveDiagram)
' Activate the new DiagramView
igxView2.View.Window.Activate
MsgBox "Click OK to view Print Preview."
igxView2.PrintPreview
MsgBox "Click OK to continue."
```

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

ScrollToObject Method

Syntax *DiagramView*.**ScrollToObject**(*DiagramObject* As DiagramObject, [*Center* As Boolean = True])

Description The ScrollToObject method scrolls the diagram view to the specified DiagramObject. Setting Center to True scrolls the diagram view so the DiagramObject is centered in the view. Setting Center to False scrolls the diagram view so the DiagramObject is in the top left corner of the view.

Example The following example adds some shapes far off screen. Then the ScrollToObject method is used to move and center the view on Shape 2.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxView1 As DiagramView
Dim igxView2 As DiagramView
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 9, 1440)
' Add a shape
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 12, 1440)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Get the current DiagramView
Set igxView1 = ActiveDiagram.Views.Item(1).DiagramView
' Add a new DiagramView
Set igxView2 = ActiveDiagram.Views.AddDiagramView(ActiveDiagram)
' Activate the new DiagramView
igxView2.View.Window.Activate
MsgBox "Click OK to scroll to Shape 2."
igxView2.ScrollToObject igxShape2.DiagramObject, True
MsgBox "Click OK to continue."
```

See Also CenterX property

CenterY property

ScrollToPage method

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

ScrollToPage Method

Syntax *DiagramView*.**ScrollToPage**(*PageNumber* As Long)

Description The ScrollToPage method scrolls the diagram view to the page number specified by the *PageNumber* argument.

Example The following example scrolls the diagram view to page 2.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxView1 As DiagramView
Dim igxView2 As DiagramView
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 9, 1440)
' Add a shape
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 12, 1440)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Get the current DiagramView
Set igxView1 = ActiveDiagram.Views.Item(1).DiagramView
' Add a new DiagramView
Set igxView2 = ActiveDiagram.Views.AddDiagramView(ActiveDiagram)
' Activate the new DiagramView
igxView2.View.Window.Activate
MsgBox "Click OK to scroll to Shape 2."
igxView2.ScrollToPage 2
MsgBox "Click OK to continue."
```

See Also ScrollToObject method

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

View Property

Syntax *DiagramView*.**View**

Data Type View object (read-only, See Object Properties)

Description The View property returns the View object that represents the specified DiagramView. The View object provides access higher level view properties such as the client Window.

Example The following example uses the View object Window property to activate a new DiagramView.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxView1 As DiagramView
Dim igxView2 As DiagramView
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 9, 1440)
' Add a shape
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 12, 1440)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Get the current DiagramView
Set igxView1 = ActiveDiagram.Views.Item(1).DiagramView
' Add a new DiagramView
Set igxView2 = ActiveDiagram.Views.AddDiagramView(ActiveDiagram)
' Activate the DiagramView
MsgBox "Click OK to activate View 2."
igxView1.View.Window.Activate
MsgBox "Click OK to continue."
```

See Also View object

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

ZoomPercentage Property

Syntax *DiagramView*.**ZoomPercentage**

Data Type Long (read/write)

Description The ZoomPercentage property specifies the zoom level for the diagram view. The value is given as a percentage. The default is 100%.

Example The following example sets the ZoomPercentage to 50%.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxView1 As DiagramView
Dim igxView2 As DiagramView
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a shape
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Get the current DiagramView
Set igxView1 = ActiveDiagram.Views.Item(1).DiagramView
' Add a new DiagramView
Set igxView2 = ActiveDiagram.Views.AddDiagramView(ActiveDiagram)
' Activate the DiagramView
MsgBox "Click OK to activate Zoom 50%."
igxView2.View.Window.Activate
' Zoom out 50%
igxView2.ZoomPercentage = 50
MsgBox "Click OK to continue."
```

See Also Width property

Height property

{button DiagramView object,JI(`igrafxrf.HLP',`DiagramView_Object')}

Shape Object

The Shape object is the programmatic object that is associated with an iGrafx Professional shape. Using this object, you can perform any action with shapes that you can through the user interface, and more.

Shape and the Object Hierarchy

The Shape relates to other objects in the following ways:

- A shape is subordinate to a diagram in the object hierarchy; that is, diagrams contain shapes (as well as other objects).
- A shape is a DiagramObject, but not the only type of DiagramObject object.
- Shapes can be found in the DiagramObjects collection of a diagram.
- The combination of the Shape and DiagramObject objects form what is known as a ‘‘composite control.’’
- A shape has a Graphic.

Generalization Issues for the Shape Object

There can be times, when working with an object like the Shape object, that you may think it is missing some critical properties; for instance, how do you position a shape? The answer is that a Shape object (and other objects within the iGrafx Professional API, too) has what is called an ‘‘Extender’’ object (see the next section); in the case of the Shape object, the extender is the DiagramObject object. Common properties that apply to all objects in the diagram are found in the DiagramObject.

The Shape Object’s Relationship with the DiagramObject

A Shape is always a DiagramObject; a DiagramObject may be a Shape. For all VBA controls that are created on VBA project items, the DiagramObject object and the Shape object together are used to create a composite control.

For example, if you click on a shape and choose Edit Code for that shape, iGrafx Professional creates a VBA control for that shape. If it is the first shape to have its code edited, the VBA control name is ‘‘Shape1’’.

If you examine the Shape1 control carefully, you will notice that it is a composite control that has all the properties, methods, and events of the Shape object **and** all the properties, methods, and events of the DiagramObject object. The DiagramObject object’s properties, methods, and events are called the ‘‘Extender’’ portion of the composite control. The Shape object’s properties, methods, and events are called the ‘‘Primary’’ portion of the composite control.

If you access a shape through means other than the VBA control (for example, by writing a statement like `ActiveDiagram.DiagramObjects.Item(1).Shape`), then you get only the ‘‘Primary’’ shape properties, methods, and events. However, observe that the shape is a property of DiagramObject (and vice versa), which gives you easy access to both the ‘‘Primary’’ and ‘‘Extender’’ properties, methods, and events that are automatically merged together when a VBA control is created.

The Shape Object Compared to the TextGraphicObject

In iGrafx Professional, a shape is more than just a graphical element. In fact, there is a distinct difference between shapes and graphics (e.g. a TextGraphicObject). A shape has the following characteristics:

- It can have VBA code associated with it.
- It can have custom data and fields associated with it.
- It has connection points so it can be connected to other shapes.
- It can have modeling data associated with it.

Additionally, in iGrafx Process a shape is also known as an activity. An activity is the union of the shape's characteristics and the additional modeling data that iGrafx Process associates with a shape.

In contrast to the Shape object, neither the TextGraphicObject object nor the Graphic object have any of these characteristics. In fact, the Graphic object is a property (and subordinate to) both the Shape object and the TextGraphicObject object.

The Shape Object's Relationship to the Graphic Object

As mentioned in the previous section, a Shape object has a Graphic object property. The Graphic object defines the visual representation of the shape.

The Shape object and the Graphic object have a number of properties in common; for instance, the fill formatting and line formatting properties. A property set at the Shape object level overrides the same property set at the Graphic object level. This is always true unless you use the ProtectFillFormat and ProtectLineFormat properties of the Graphic object. These two properties specify that the fill and line format settings at the Graphic level should not be overridden by property settings at the Shape level.

For example, assume you have a shape whose Graphic object consists of two rectangles with a solid green fill. If you set the fill format at the Shape object level to be a red gradient, then both rectangles in the graphic are filled with that red gradient rather than the solid green, unless you set the ProtectFillFormat property to True.

Creating a Shape

The following example creates a new shape object on the active diagram. As the new shape is created, it is set to the igxShape variable.

```
' Dimension the variables
Dim igxShape As Shape
' Create a new shape with its center at X = 1, Y = 1, and
' set it to the igxShape variable
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
  (1440, 1440, ActiveDiagram.DiagramType.ShapeLibrary.Item(1))
```

Properties, Methods, and Events

All of the properties, methods, and events for the Shape object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Adjustments	ConvertToGraphic	AdjustmentMove
Application	FitToText	AfterAdjustmentMove
AutoGrow	Replace	AfterConnectorAttach
BackColor	RevertToShapeClass	AfterConnectorDetach
BottomDepartment		BeforeAdjustmentMove
DecisionCases		BeforeConnectorAttach
DepartmentRange		BeforeConnectorDetach
DiagramObject		BeforeExecuteLink
ExcludedDepartmentNames		BeforeReplace
FillColor		ChangeDepartment
FillType		EntitiesAbort
GradientIndex		EntitiesFinished
Graphic		EntitiesStart
InputConnectorLines		EntityAccept
IsCrossDepartment		EntityExecute
IsDecision		EntityInitiate
IsStartPoint		EntityLeave
LineColor		EntityStep
LineStyle		SetLink
LineWidth		

Links
Note
OutputConnectorLines
OutputPaths
Parent
PatternIndex
PermanentShape
ShadowColor
ShadowDepth
ShadowType
ShapeClass
ShapeFormat
ShapeNumber
ShowNumbering
StartPointName
Text
TextBlock
TextLF
TextRTF
ThreeDDepth
ThreeDType
TopDepartment

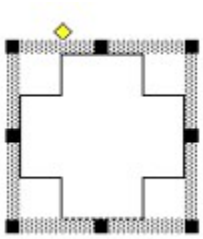
Related Topics

ShapeLibraries object
ShapeLibrary object
ShapeLibraryItem object
ShapeClass object
ShapeNumber object
ShapeFormat object
DiagramObject object
iGrafx API Object Hierarchy

AdjustmentMove Event

Syntax **Private Sub** *Shape*_**AdjustmentMove**(*Index* As Integer, *X* As Double, *Y* As Double)

Description The AdjustmentMove event occurs when an Adjustment object associated with the specified shape is moved. A shape with an adjustment may look something like the following diagram:



The yellow diamond is an adjustment. For this shape, dragging the adjustment from left to right changes the thickness of the cross shape.

The event parameters provide the following data:

- The *Index* parameter is the index number within the Adjustments collection of the adjustment point being moved. You can use this argument to handle each adjustment point associated with a shape differently.
- The *X* parameter is the horizontal position of the adjustment. The units of measure for this value are in shape coordinate space. The *X* parameter is passed by reference, so if you change the value of X, it affects the X position of the adjustment.
- The *Y* parameter indicates the vertical position of the adjustment. The units of measure for this value are in shape coordinate space. The *Y* parameter is passed by reference, so if you change the value of Y, it affects the Y position of the adjustment.

A Shape object also has a BeforeAdjustmentMove and an AfterAdjustmentMove event. The difference is that these events are fired once—the BeforeAdjustmentMove event is fired when the user first clicks on the adjustment. The AfterAdjustmentMove event is fired when the user stops dragging the adjustment and releases the mouse button. The AdjustmentMove event is fired continually as the adjustment point is moved.

Typically, you use the AdjustmentMove events to allow a change in the position of an Adjustment to change the graphic of the shape in some way. You can also associate a change in the position of an Adjustment with some data associated with the shape.

If the change you are making to the shape requires a lot of computations, you might consider using the BeforeAdjustmentMove and AfterAdjustmentMove events instead of the AdjustmentMove event. Since the AdjustmentMove event could be fired hundreds of time during the course of an adjustment being dragged, if the code you write in this event does not execute quickly, the user may experience significant slowdowns while dragging an adjustment.

Example The following example illustrates a simple shape consisting of a square with an inset star. The GraphicBuilder object is used to draw the square and the inset star. With the Adjustment object, a user can control the position of the inset star by dragging the adjustment point. The following picture illustrates two possible positions of the star:

Copy this code into your Diagram or Process project code window. The Test( ) subroutine adds the parallelogram iShape, which has one adjustment point, to the diagram. The graphic of this shape is replaced in the AdjustmentMove event.

```
Public Sub Test()
    Dim igxShapeLib As ShapeLibrary
    Dim igxShape As Shape
    ' Add the iShapes library (iShapes have adjustment points)
    Set igxShapeLib = Application.ShapeLibraries.Add _
        ("Intelligent Shapes", "Basic iShapes")
    ' Add the Parallelogram iShape to the diagram
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440, igxShapeLib.Item(1))
    ' Display a message that the event is ready to test
    MsgBox "iShape created. The event is now active." _
        & Chr(13) & "Return to the diagram and try draging " _
        & "the adjustment point."
End Sub

' This event is fired continuouly as an Adjustment Point is
' moved on an iShape
Private Sub AnyShape_AdjustmentMove(ByVal Index As Integer, X As Double, Y As
Double)
    ' Restrict the area that the adjustment can be dragged
    ' within the shape by changing out of bounds X and Y values
    If X < 0.3 Then X = 0.3
    If X > 0.7 Then X = 0.7
    If Y < 0.3 Then Y = 0.3
    If Y > 0.7 Then Y = 0.7
    ' The GraphicBuilder object is used to draw a star and a rectangle
    Dim igxGraphicBuilder As New GraphicBuilder
    igxGraphicBuilder.BeginPath
    igxGraphicBuilder.Star X, Y, 0.3, 0.1, 10
    igxGraphicBuilder.Rectangle 0, 0, 1, 1
    igxGraphicBuilder.EndPath
    ' Replace the shape's graphic with the one made with
    ' the GraphicBuilder object
    AnyShape.Graphic.Replace igxGraphicBuilder.Graphic
End Sub
```

The following example changes the fill color of the shape based on the value of the X position of the adjustment. Also, by setting the value of the Y position to 0, then the adjustment point is allowed to move horizontally only. This is because the new Y position is not applied to the adjustment; instead, a value of zero is applied. You can use any value from 0.0 to 1.0 for the Y position. If you want the adjustment point to be allowed to move vertically only, then set the X position to a fixed value from 0.0 to 1.0. The FillColor of the shape is not changed until the left mouse button is released from the adjustment. This is because a repaint for the shape is not invoked until the mouse button is released.

Copy this code into your Diagram or Process project code window. The Test( ) subroutine adds one iShape with adjustment points to your diagram.

```
Public Sub Test()
    Dim igxShapeLib As ShapeLibrary
    ' Add the iShapes library (iShapes have adjustment points)
    Set igxShapeLib = Application.ShapeLibraries.Add _
        ("Intelligent Shapes", "Basic iShapes")
    ' Add the Parallelogram iShape to the diagram
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440, igxShapeLib.Item(1))
    ' Display a message that the event is ready to test
    MsgBox "iShape created. The event is now active." _
        & Chr(13) & _
        "Return to the diagram and try draging the adjustment point."
End Sub

Private Sub AnyShape_AdjustmentMove(ByVal Index As Integer, X As Double, Y As
Double)
    If Index = 1 Then
        AnyShape.FillColor = Int((X * 2) * 100)
        Y = 0
    End If
End Sub
```

See Also

AfterAdjustmentMove event

BeforeAdjustmentMove event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

Adjustments Property

Syntax *Shape*.**Adjustments**

Data Type Adjustments collection object (read-only, See Object Properties)

Description The Adjustments property returns the Adjustments collection for the specified Shape object. The Adjustments collection can be used to add, delete, or move the adjustment points on a shape.

Some shapes, such as iShapes, have built-in adjustment points that affect the appearance of the shape when moved, but adjustment points are not limited to that purpose. Other shapes have no built in adjustment points, but points can be added to them. Adjustment points can serve any purpose the programmer chooses by writing code to implement them.

Example The following example creates a shape on the active diagram, and then adds an adjustment point to the shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxAdjustments As Adjustments
' Create a new shape with its center at one inch and then
' set it to the igxShape variable
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, ActiveDiagram.DiagramType.ShapeLibrary.Item(1))
' Select the shape to see the adjustment points
ActiveDiagram.Selection.Add igxShape.DiagramObject
' Set the igxAdjustments variable to Adjustments object
Set igxAdjustments = igxShape.Adjustments
' Add an adjustment point to the shape at the top and center
MsgBox "Click OK to add an Adjustment Point"
igxAdjustments.Add 0.5, 0
MsgBox "Click OK to continue"
```

See Also Adjustment object

Adjustments object

iGrafx API Object Hierarchy

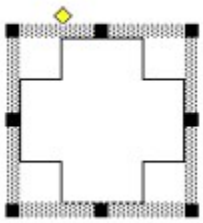
{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

AfterAdjustmentMove Event

Syntax **Private Sub** *Shape*_**AfterAdjustmentMove**(*Index* As Integer, *X* As Double, *Y* As Double)

Description The AfterAdjustmentMove event occurs after an Adjustment associated with the specified Shape object is moved (or more specifically, when the user releases the mouse button after dragging an adjustment). This event can be used to perform some action based on the new position of the adjustment.

A shape with an adjustment may look something like the following diagram:



The yellow diamond is an adjustment. For this shape, dragging the adjustment from left to right changes the thickness of the cross shape.

The event parameters provide the following data:

- The *Index* parameter is the index number within the Adjustments collection of the adjustment point that was moved. You can use this argument to handle each adjustment point associated with a shape differently.
- The *X* parameter is the horizontal position of the adjustment. The units of measure for this value are in shape coordinate space. The *X* parameter is passed by reference, so if you change the value of X, it affects the X position of the adjustment.
- The *Y* parameter indicates the vertical position of the adjustment. The units of measure for this value are in shape coordinate space. The *Y* parameter is passed by reference, so if you change the value of Y, it affects the Y position of the adjustment.

A Shape object also has a BeforeAdjustmentMove and AdjustmentMove event. The BeforeAdjustmentMove event is fired when the user first clicks on the adjustment. The AdjustmentMove event is fired continually as the adjustment point is moved.

Typically you use the AdjustmentMove events to allow a change in the position of an Adjustment to change the graphic of the shape in some way. You can also associated a change in the position of an Adjustment with some data associated with the shape.

If the change you are making to the shape requires a lot of computations, you might consider using the BeforeAdjustmentMove and AfterAdjustmentMove events instead of the AdjustmentMove event. Since the AdjustmentMove event could be fired hundreds of time during the course of an adjustment being dragged, if the code you write in this event does not execute quickly, the user can experience significant slowdowns while dragging an adjustment.

Example The following example displays a different color based on the X position of the adjustment point that is the first object in the Adjustments collection.

```
Public Sub Test()
   Dim igxShapeLib As ShapeLibrary
    Dim igxShape As Shape
    ' Add the iShapes library (iShapes have adjustment points)
    Set igxShapeLib = _
        Application.ShapeLibraries.Add("Intelligent Shapes", _
        "Basic iShapes")
    ' Add the Parallelogram iShape to the diagram
```

```
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440, igxShapeLib.Item(1))
    ' Display a message that the event is ready to test
    MsgBox "iShape created. The event is now active." & Chr(13) _
        & "Return to the diagram and try draging the adjustment point."
End Sub

Private Sub AnyShape_AfterAdjustmentMove(ByVal Index As Integer, X As Double,
Y As Double)
    If Index = 1 Then
        Select Case X
        Case Is <= 0.2
            AnyShape.FillColor = vbRed
        Case Is <= 0.4
            AnyShape.FillColor = vbGreen
        Case Is <= 0.6
            AnyShape.FillColor = vbBlue
        Case Is <= 0.8
            AnyShape.FillColor = vbYellow
        Case Is <= 1
            AnyShape.FillColor = vbBlack
        End Select
    End If
End Sub
```

See Also

AdjustmentMove event

BeforeAdjustmentMove event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

AfterConnectorAttach Event

Syntax **Private Sub** *Shape*_**AfterConnectorAttach**(*Connector* As ConnectorLine, ByVal *Source* As Boolean)

Description The AfterConnectorAttach event occurs after a connector line is attached to the specified Shape object. The event parameters provide the following data:

- The *Connector* parameter returns a ConnectorLine object. This is the connector that has just been connected to a shape.
- The *Source* parameter returns a Boolean value that indicates whether the shape being attached to is the source of the connector line. If *Source* is False, then the shape is the destination of the connector. If *Source* is True, then the shape is the source of the connector. The source is the object from which the connector is drawn. The destination is the object to which the connector is drawn.

Example The following example shows an AfterConnectorAttach event based on the AnyShape object. The code for this event affects the active diagram. When any shape in the active diagram has a connector line connected to it, the color of the source shape is changed to green, the color of the destination shape is changed to red, and the connector line is changed to yellow.

```
Public Sub MakeShapes()
   Dim igxShape As Shape
   ' Add two shapes to the active diagram
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape 1440, 1440
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape 1440, 1440 * 3
   ' Display a message that the event is ready
   MsgBox "Shapes created. The event is now active." _
       & Chr(13) & _
       "Return to the diagram and try dragging a connector" _
       & Chr(13) & "Line between the shapes."
End Sub

Private Sub AnyShape_AfterConnectorAttach(Connector As ConnectorLine, Source
As Boolean)
    If Source Then
        AnyShape.FillColor = vbGreen
        Connector.LineColor = vbYellow
    Else
        AnyShape.FillColor = vbRed
        Connector.LineColor = vbYellow
    End If
End Sub
```

See Also AfterConnectorDetach event

BeforeConnectorAttach event

BeforeConnectorDetach event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

AfterConnectorDetach Event

Syntax **Private Sub** *Shape*_**AfterConnectorDetach**(*Connector* As ConnectorLine, ByVal *Source* As Boolean)

Description The AfterConnectorDetach event occurs after a connector line is detached from a Shape object. The event parameters provide the following data:

- The *Connector* parameter returns a ConnectorLine object. This is the connector that has just been detached from the shape.
- The *Source* parameter returns a Boolean value that indicates whether the shape being detached from is the source of the connector line. If *Source* is False, then the shape is the destination of the connector. If *Source* is True, then the shape is the source of the connector. The source is the object from which the connector is drawn. The destination is the object to which the connector is drawn.

Example The following example uses the AnyShape object to turn any shape or connector line green when connected, or white when detached.

```
Public Sub MakeShapes()
    Dim igxShape As Shape
    ' Add two shapes to the diagram
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape 1440, 1440
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440 * 2)
    ' Display a message that the event is ready
    MsgBox "Both events are now active. Try connecting and" _
        & Chr(13) & "disconnecting the connector lines from " _
        & "the two shapes" & Chr(13) _
        & "in various combinations, and observe the behavior."
End Sub

Private Sub AnyShape_AfterConnectorAttach(ByVal Connector As ConnectorLine,
ByVal Source As Boolean)
    AnyShape.FillColor = vbGreen
    Connector.LineColor = vbGreen
End Sub

Private Sub AnyShape_AfterConnectorDetach(ByVal Connector As ConnectorLine,
ByVal Source As Boolean)
    AnyShape.FillColor = vbWhite
    Connector.LineColor = vbBlack
End Sub
```

See Also AfterConnectorAttach event

BeforeConnectorDetach event

BeforeConnectorAttach event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

AutoGrow Property

Syntax *Shape*.**AutoGrow**[= {**True | False**}]

Data Type Boolean (read/write)

Description The AutoGrow property specifies whether a shape automatically resizes when text is added to the shape. If you want a shape to increase in size to accommodate text added to it, set AutoGrow to True. If you want a shape to remain a fixed size regardless of how much text is added to it, set AutoGrow to False.

Example The following example creates two shapes—one with AutoGrow turned on, and one with AutoGrow turned off. The same text is then added to both shapes in order to compare how the Autogrow property affects the shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxDiagObjects As DiagramObjects
' Get the DiagramObjects object
Set igxDiagObjects = ActiveDiagram.DiagramObjects
' Create 2 new shapes in the Diagram
Set igxShape1 = igxDiagObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1)(5))
Set igxShape2 = igxDiagObjects.AddShape _
    (1440, 1440 * 3, Application.ShapeLibraries.Item(1)(5))
igxShape1.Text = "AutoGrow TRUE"
igxShape2.Text = "AutoGrow FALSE"
' Set the AutoGrow property of each shape
igxShape1.AutoGrow = True
igxShape2.AutoGrow = False
' Add a large amount of text to each shape
MsgBox "Click OK to add text to each shape."
igxShape1.Text = "This shape has AutoGrow set to True. It has" _
    & " expanded to fit this large amount of text."
igxShape2.Text = "This shape has AutoGrow set to False. It has" _
    & " not expanded to fit this large amount of text."
MsgBox "Click OK to continue."
```

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

BackColor Property

Syntax *Shape*.**BackColor**

Data Type Color (read/write)

Description The BackColor property specifies the background color for the Shape object. Color values are specified with the RGB function, or with one of the VB color constants.

Certain settings of the FillType and LineStyle properties affect how the background color is used. The following list describes these specific situations.

- If the FillType property is set to IxFillNone or IxFillSolid, the background color has no effect.
- If the FillType property is set to IxFillPattern, the background color is used as the background behind the pattern. Refer to the Format—Fill dialog in the iGrafx Professional user interface, or to the iGrafx Professional User's Guide for more information about fill patterns.
- If the FillType property is set to IxFillGradient, the background color is used as the EndColor in the gradient style (the FillColor is used as the StartColor). Refer to the Format—Fill dialog in the iGrafx Professional user interface, or to the iGrafx Professional User's Guide for more information about gradients.
- If the LineStyle property is set to any of the broken line styles (dashed, dotted, etc.), the background color is used to fill the gaps in the broken line. This allows you to preserve your fill color independent of the lines used for the outline of the graphic.

This functionality is also contained in the ShapeFormat object (see the ShapeFormat property), which allows you to set a shape's formats for line and fill types, and shadow and 3D effects.

Example The following example creates a shape on the active diagram, and then sets the fill type to gradient. The gradient's colors are then set to blue for the fill color and green for the back color.

```
' Dimension the variables
Dim igxShape As Shape
' Create a new Shape object on the active diagram.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the FillType Property of the shape that was added
igxShape.FillType = ixFillGradient
' Set the angle and direction for the gradient by index
igxShape.GradientIndex = 5
' Set the start color (FillColor) for the gradient to blue
igxShape.FillColor = vbBlue
' Set the end color (BackColor) for the gradient to green
igxShape.BackColor = vbGreen
```

See Also ShapeFormat property

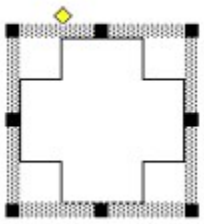
{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

BeforeAdjustmentMove Event

Syntax **Private Sub** *Shape*_**BeforeAdjustmentMove**(*Index* As Integer*, X* As Double*, Y* As Double)

Description The BeforeAdjustmentMove event occurs when an Adjustment associated with a shape is first moved (or more specifically, when the user first clicks down on the adjustment with the mouse). This event can be used to perform some action based on the original position of the adjustment. For example, the values could be stored in global variables and then used with the AdjustmentMove event to reset the original X and Y positions of the adjustment.

A shape with an adjustment may look something like the following diagram:



The yellow diamond is an adjustment. For this shape, dragging the adjustment from left to right changes the thickness of the cross shape.

The event parameters provide the following data:

- The *Index* parameter is the index number within the Adjustments collection of the adjustment point to be moved. You can use this argument to handle each adjustment point associated with a shape differently.
- The *X* parameter is the horizontal position of the adjustment. The units of measure for this value are in shape coordinate space. The *X* parameter is passed by reference, so if you change the value of X, it affects the X position of the adjustment.
- The *Y* parameter indicates the vertical position of the adjustment. The units of measure for this value are in shape coordinate space. The *Y* parameter is passed by reference, so if you change the value of Y, it affects the Y position of the adjustment.

A Shape object also has an AfterAdjusmtentMove and AdjustmentMove event. The AfterAdjustmentMove event is fired when the user releases the mouse button after dragging an adjustment. The AdjustmentMove event is fired continually as the adjustment point is moved.

Typically you use the AdjustmentMove events to allow a change in the position of an Adjustment to change the graphic of the shape in some way. You can also associated a change in the position of an Adjustment with some data associated with the shape.

If the change you are making to the shape requires a lot of computations, you might consider using the BeforeAdjustmentMove and AfterAdjustmentMove events instead of the AdjustmentMove event. Since the AdjustmentMove event could be fired hundreds of time during the course of an adjustment being dragged, if the code you write in this event does not execute quickly, the user can experience significant slowdowns while dragging an adjustment.

Example The following example outputs the starting X and Y values for the adjustment point on a shape to the Output window when the adjustment point is moved.

```
Public Sub Test()
    ' Dimension the variables
   Dim igxShapeLib As ShapeLibrary
   Dim igxShape As Shape
   ' Add the iShapes library (iShapes have adjustment points)
   Set igxShapeLib = _
       Application.ShapeLibraries.Add("Intelligent Shapes", _
```

```
        "Basic iShapes")
    ' Add the Parallelogram iShape to the diagram
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440, _
        igxShapeLib.Item(1))
    ' Select the shape to show the adjustment point
    ActiveDiagram.Selection.Add igxShape.DiagramObject
    ' Display message that the event is ready to test
    MsgBox "Shape created. The event is now active." _
        & Chr(13) & _
        "Return to the diagram and try dragging the adjustment point."
End Sub

Private Sub AnyShape_BeforeAdjustmentMove(ByVal Index As Integer, X As Double,
Y As Double)
    ' Display the position of the adjustment point in the output window
    Output "Point #" & Index & ", X:" & X & ", Y:" & Y
End Sub
```

See Also AdjustmentMove event

AfterAdjustmentMove event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

BeforeConnectorAttach Event

Syntax **Private Sub** *Shape*_**BeforeConnectorAttach**(*Connector* As ConnectorLine, ByVal *Source* As Boolean, *Cancel* As Boolean)

Description The BeforeConnectorAttach event occurs before a connector line is attached to the specified Shape object. You can set the *Cancel* parameter to True to prevent the connector line from being attached to the shape.

The event parameters provide the following data:

- The *Connector* parameter returns a ConnectorLine object. This is the connector line that is about to be attached to the shape.
- The *Source* parameter returns a Boolean value that indicates whether the shape being attached to is the source of the connector line. If *Source* is False, then the shape is the destination of the connector. If *Source* is True, then the shape is the source of the connector. The source is the object from which the connector is drawn. The destination is the object to which the connector is drawn.
- The *Cancel* parameter, when set to True, prevents the connector line from being attached to the shape.

Example The following example uses the BeforeConnectorAttach event to prevent the connector line from being connected into "Process" shapes.

```
Public Sub Test()
   Dim igxShapeLib As ShapeLibrary
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Add the iShapes library (iShapes have adjustment points)
   Set igxShapeLib = Application.ShapeLibraries.Add _
       ("Intelligent Shapes", "Basic iShapes")
   ' Add two iShapes to the diagram
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440, _
       igxShapeLib.Item(1))
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 5, igxShapeLib.Item(2))
   ' Display a message that the event is ready to test
   MsgBox "Shapes created. The event is now active. Return to the" _
       & Chr(13) & _
       "diagram and try connecting the shapes using connector lines."
End Sub

Private Sub AnyShape_BeforeConnectorAttach(ByVal Connector As ConnectorLine,
ByVal Source As Boolean, Cancel As Boolean)
    ' Prevent connectors from connecting to rectangles
   If AnyShape.DiagramObject.ObjectName = "Rectangle" Then
       MsgBox "Not allowed to add connector lines to this diagram"
       Cancel = True
       Connector.Delete
   End If
End Sub
```

See Also AfterConnectorAttach event

AfterConnectorDetach event

BeforeConnectorDetach event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

BeforeConnectorDetach Event

Syntax **Private Sub** *Shape*_**BeforeConnectorDetach**(*Connector* As ConnectorLine, ByVal *Source* As Boolean, *Cancel* As Boolean)

Description The BeforeConnectorDetach event occurs before a connector line is detached from the specified Shape object. You can set the *Cancel* parameter to True to prevent the connector line from being detached from the shape.

The event parameters provide the following data:

- The *Connector* parameter returns a ConnectorLine object. This is the connector line that is about to be detached from the shape.
- The *Source* parameter returns a Boolean value that indicates whether the shape being detached from is the source of the connector line. If *Source* is False, then the shape is the destination of the connector. If *Source* is True, then the shape is the source of the connector. The source is the object from which the connector is drawn. The destination is the object to which the connector is drawn.
- The *Cancel* parameter, when set to True, prevents the connector line from being detached from the shape.

Example The following example uses the BeforeConnectorDetach event to prevent the connector line from being detached from the shapes.

```
Public Sub Test()
   Dim igxShapeLib As ShapeLibrary
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Add the iShapes library (iShapes have adjustment points)
   Set igxShapeLib = Application.ShapeLibraries.Add _
       ("Intelligent Shapes", "Basic iShapes")
   ' Add two iShapes to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440, _
       igxShapeLib.Item(1))
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 5, igxShapeLib.Item(2))
    ' Connect the shapes
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
       ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
       ixDirWest, ixConnectRelativeToShape)
   ' Display a message that the event is ready to test
   MsgBox "Shapes created. The event is now active. Return to the" _
       & Chr(13) & _
       "diagram and try detaching the connector line."
End Sub

Private Sub AnyShape_BeforeConnectorDetach(ByVal Connector As
IGrafx2.IXConnector, ByVal Source As Boolean, Cancel As Boolean)
    Cancel = True
    MsgBox "Detaching a connector is not permitted"
End Sub
```

See Also AfterConnectorDetach event

AfterConnectorAttach event
BeforeConnectorAttach event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

BeforeExecuteLink Event

Syntax **Private Sub** *Shape*_**BeforeExecuteLink**(*Link* As Link, *Cancel* As Boolean)

Description The BeforeExecuteLink occurs before a link is executed or jumped to. You can prevent the execution of the link by setting the *Cancel* parameter to True. You might use this event to check if a link is valid or to modify or change a link in some way before iGrafx Professional jumps to or executes the link.

The event parameters provide the following data:

- The *Link* parameter returns a Link object. This is the link that is about to be executed.
- The *Cancel* parameter, when set to True, prevents the link from being executed.

Example The following example displays a dialog box asking the user to confirm the execution of a link.

```
Public Sub Test()
   Dim igxShapeLib As ShapeLibrary
   Dim igxShape1 As Shape
   ' Add the iShapes library (iShapes have adjustment points)
   Set igxShapeLib = Application.ShapeLibraries.Add _
       ("Intelligent Shapes", "Basic iShapes")
   ' Add the Parallelogram iShape to the diagram
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440, 1440, igxShapeLib.Item(1))
   ' Give the shape a file link to Notepad
   igxShape1.Links.AddFileLink "C:\Winnt\notepad.exe"
   ' Display a message that the event is ready to test
   MsgBox "Shape and link created. The BeforeExecuteLink event is " _
       & "now active." & Chr(13) & _
       "Return to the diagram and try excuting the link" _
       & Chr(13) & "(RightClick the shape, then select the link.)"
End Sub

Private Sub AnyShape_BeforeExecuteLink(ByVal Link As Link, Cancel As Boolean)
   If (MsgBox("Execute WordPad?", vbYesNo) = vbNo) Then
       Cancel = True
   End If
End Sub
```

See Also Link object

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

BeforeReplace Event

Syntax **Private Sub** *Shape*_**BeforeReplace**(*Cancel* As Boolean)

Description The BeforeReplace event occurs before the specified Shape object is replaced. You can prevent the shape from being replaced by setting the *Cancel* parameter to True. A shape can be replaced by using either of the following:

- The Shape.Replace method from the iGrafx Professional API
- The Arrange, Replace Shape… command from the iGrafx Professional user interface.

Example The following example displays a dialog box asking the user to confirm before replacing any shape.

```
Public Sub Test()
    ' Dimension the variables
    Dim igxShape As Shape
    ' Add a shape to the diagram
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    MsgBox "Shape created. Click OK to Replace it"
    ' Replace the shape with a ShapeLibrary item
    ' This fires the event
    igxShape.Replace ShapeLibraries.Item(1).Item(1)
End Sub

Private Sub AnyShape_BeforeReplace(Cancel As Boolean)
    If MsgBox("Replace shape?", vbYesNo) = vbNo Then
        Cancel = True
    End If
End Sub
```

See Also Replace method

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

BottomDepartment Property

Syntax *Shape*.**BottomDepartment**

Data Type Department object (read-only, See Object Properties)

Description The BottomDepartment property returns the Department object that is the last (or bottom-most) department on the diagram to which the specified shape belongs. When a shape is placed in a diagram within the boundaries of a department, the TopDepartment and BottomDepartment properties are filled in automatically. The property returns an object that contains Nothing if there is no bottom department for the shape.

Setting the TopDepartment and the BottomDepartment properties causes iGrafx Professional to stretch the shape from the top of the TopDepartment to the bottom of the BottomDepartment. All departments that a shape is drawn in are listed in the DepartmentRange collection for the shape. This information, along with the shape's ExcludedDepartmentNames property, position a shape relative to the departments in a diagram.

Example The following example creates three departments, and then places a shape so it is located within the boundaries of the first department. It then tests to determine whether the TopDepartment and BottomDepartment properties were set automatically when the shape was created. It then changes the value of each property to show that the shape expands and moves based on the value of these properties.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDepartment1 As Department
Dim igxDepartment2 As Department
Dim igxDepartment3 As Department
' Create three departments
Set igxDepartment1 = _
    ActiveDiagram.Departments.AddDepartment("TestDept1")
Set igxDepartment2 = _
    ActiveDiagram.Departments.AddDepartment("TestDept2")
Set igxDepartment3 = _
    ActiveDiagram.Departments.AddDepartment("TestDept3")
' Create a new shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 1.5, 1440)
' Test whether placement of the shape sets the top and bottom
' department properties for the shape
If (igxShape.TopDepartment = "Nothing") Then
    MsgBox "Top department property is not set." & Chr(13) _
        & "Set it to TestDept1."
    igxShape.TopDepartment = igxDepartment1
Else
    MsgBox "Top department property was set on placement. " _
        & "It is: " & igxShape.TopDepartment.DepartmentName
End If
If (igxShape.BottomDepartment = "Nothing") Then
    MsgBox "Bottom department property is not set." & Chr(13) _
        & "Set it to TestDept1."
    igxShape.BottomDepartment = igxDepartment1
Else
    MsgBox "Bottom department property was set on placement. " _
        & "It is: " & igxShape.BottomDepartment.DepartmentName
End If
```

```
' Change the top and bottom departments for the shape
igxShape.BottomDepartment = igxDepartment3
MsgBox "View the result."
igxShape.TopDepartment = igxDepartment2
MsgBox "View the result."
```

See Also

DepartmentRange property

ExcludedDepartmentNames property

TopDepartment property

Department object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ChangeDepartment Event

Syntax **Private Sub** *Shape*_**ChangeDepartment**()

Description The ChangeDepartment event occurs when any department association is changed for the specified Shape object. For example, if a shape is associated with two departments and one of those associations is changed or removed, this event is triggered. A change to a department association can occur in many ways, including moving the shape, changing the excluded department names list, etc.

Example The following example implements the ChangeDepartment event. When the user moves a shape into another department, or spans new departments, the event displays the departments that the shape now occupies.

```
Public Sub Test()
    ' Dimension the variables
   Dim igxShape As Shape
   Dim igxDepartment1 As Department
   Dim igxDepartment2 As Department
   Dim igxDepartment3 As Department
   ' Create three departments
   Set igxDepartment1 = _
       ActiveDiagram.Departments.AddDepartment("TestDept1")
   Set igxDepartment2 = _
       ActiveDiagram.Departments.AddDepartment("TestDept2")
   Set igxDepartment3 = _
        ActiveDiagram.Departments.AddDepartment("TestDept3")
   ' Create a new shape
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440)
   ' Set the top and bottom departments for the shape
   igxShape.TopDepartment = igxDepartment1
   igxShape.BottomDepartment = igxDepartment3
   igxShape.ExcludedDepartmentNames.Add "TestDept2"
   MsgBox "Several Departments and a Shape created. Return to " _
       & "the diagram " & Chr(13) & "and try dragging and " _
       & "spanning the shape into other departments." _
       & Chr(13) & "Span a shape with Ctrl+Click on a shape's " _
       & "corner handle."
End Sub

Private Sub AnyShape_ChangeDepartment()
    ' Dimension the variables
   Dim sDeptNames As String
   Dim Index As Integer
   ' Collect the shapes department(s) into a string
   For Index = 1 To AnyShape.DepartmentRange.Count
       sDeptNames = sDeptNames + AnyShape.DepartmentRange.Item _
           (Index) + Chr(13)
   Next Index
   ' Display the string in the output window
   MsgBox "That shape occupies department(s):" & Chr(13) _
       & Chr(13) & sDeptNames
End Sub
```

See Also

DepartmentRange property

ExcludedDepartmentNames property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ConvertToGraphic Method

Syntax *Shape*.**ConvertToGraphic** As TextGraphicObject

Description The ConvertToGraphic method converts the specified Shape object into a TextGraphicObject object. When a shape is converted, it loses all of the attributes of a shape, such as connection points, property lists, etc. The result of the method must be assigned to a variable of type TextGraphicObject.

Example The following example creates a shape, and then converts it to a TextGraphicObject object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextGraphic As TextGraphicObject
' Set the igxShape variable to a new Shape object.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Convert the shape to a graphic
MsgBox "Click OK to convert the shape to a graphic."
Set igxTextGraphic = igxShape.ConvertToGraphic()
MsgBox "Click OK to continue."
```

See Also TextGraphicObject.ConvertToShape method

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

DecisionCases Property

Syntax *Shape*.**DecisionCases**

Data Type DecisionCases collection object (read-only, See Object Properties)

Description The DecisionCases property returns the DecisionCases collection for the specified Shape object. See the iGrafx Professional User's Guide for more information on decision cases.

Example The following example creates a shape on the active diagram, and then adds two decision cases to the shape. It then adds two more shapes to the active diagram and then connects the first shape into the two other shapes. After the shapes are connected, the decision case text is added to the two connector line paths.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxDecisionCases As DecisionCases
' Create the first shape on the active diagram.
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the DecisionCases collection
Set igxDecisionCases = igxShape1.DecisionCases
' Add a decision case to the DecisionCases collection
igxDecisionCases.Add "Path 1"
' Add a decision case to the DecisionCases collection
igxDecisionCases.Add "Path 2"
' Create the second shape on the active diagram
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
' Create the third shape on the active diagram
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 5)
MsgBox "Click OK to add connectors"
' Connect Shape 1 to Shape 2
ActiveDiagram.DiagramObjects.AddConnectorLine ixRouteRightAngle, , _
    igxShape1, ixDirSouth, , , , igxShape2, ixDirNorth
' Connect Shape 1 to Shape 3
ActiveDiagram.DiagramObjects.AddConnectorLine ixRouteRightAngle, , _
    igxShape1, ixDirEast, , , , igxShape3, ixDirNorth
MsgBox "DecisionCase labels now visible. Click OK to continue"
```

See Also DecisionCase object

DecisionCases object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

DepartmentRange Property

Syntax *Shape*.**DepartmentRange**

Data Type DepartmentRange object (read-only, See Object Properties)

Description The DepartmentRange property returns the DepartmentRange object for the specified Shape object. The DepartmentRange collection contains the list of all departments that the shape is associated with. You can use the DepartmentRange object to add and remove a shape's department associations.

Example The following example implements the ChangeDepartment event to display a shape's DepartmentRange. When the user moves a shape into another department, or spans new departments, the event displays the departments that the shape now occupies.

```
Public Sub Test()
    ' Dimension the variables
    Dim igxShape As Shape
    Dim igxDepartment1 As Department
    Dim igxDepartment2 As Department
    Dim igxDepartment3 As Department
    ' Create three departments
    Set igxDepartment1 = _
        ActiveDiagram.Departments.AddDepartment("TestDept1")
    Set igxDepartment2 = _
        ActiveDiagram.Departments.AddDepartment("TestDept2")
    Set igxDepartment3 = _
        ActiveDiagram.Departments.AddDepartment("TestDept3")
    ' Create a new shape
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape(2500, 500)
    ' Set the top and bottom departments for the shape
    igxShape.TopDepartment = igxDepartment1
    igxShape.BottomDepartment = igxDepartment3
    igxShape.ExcludedDepartmentNames.Add "TestDept2"
    MsgBox "Several Departments and a Shape created. Return to " _
        & "the diagram" & Chr(13) & "and try dragging and " _
        & "spanning the shape into other departments." _
        & Chr(13) & "Span a shape with Ctrl+Click on a shape's " _
        & "corner handle."
End Sub

Private Sub AnyShape_ChangeDepartment()
    ' Dimension the variables
    Dim sDeptNames As String
    Dim Index As Integer
    ' Collect the shapes department(s) into a string
    For Index = 1 To AnyShape.DepartmentRange.Count
        sDeptNames = sDeptNames + AnyShape.DepartmentRange.Item _
            (Index) & Chr(13)
    Next Index
    ' Display the string in the output window
    MsgBox "That shape occupies department(s):" & Chr(13) _
        & Chr(13) & sDeptNames
End Sub
```

See Also

DepartmentRange object

ExcludedDepartmentNames object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

DiagramObject Property

Syntax *Shape*.**DiagramObject**

Data Type DiagramObject object (read-only, See Object Properties)

Description A Shape is a DiagramObject, as discussed in the Shape object topic. The DiagramObject property returns the Shape object's "Extender", which is the DiagramObject object associated with the shape. Several properties and methods that are common to all objects in the diagram are at the DiagramObject level, for example positioning properties.

If you are familiar with object-oriented terminology, you can think of the DiagramObject as the base class for the Shape object (and the base class for other objects including ConnectorLine, TextGraphic, Department, and OleObject).

Example The following example creates a shape on the active diagram, and then retrieves the DiagramObject object of the shape. Using the DiagramObject object, the name of the shape is displayed in a message box.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDiagramObject As DiagramObject
' Create a shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the DiagramObject object of the new shape
Set igxDiagramObject = igxShape.DiagramObject
' Display the name of the shape
MsgBox "Shape.DiagramObject.ObjectName is " _
    & igxDiagramObject.ObjectName
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

EntitiesAbort Event

Syntax **Private Sub** *Shape*_**EntitiesAbort**(ByVal *Error* As Long)

Description The EntitiesAbort event occurs when an iDiagram is stopped before completing. This event is triggered by the following:

- Any error that occurs while the entities are running.
- Pressing the Esc key (or the programmer sending the Esc key).
- The user or programmer issuing the Stop command (through the Entity manager in the user interface or with the Document.Stop method or the Entity.Stop method).

This event can be useful if you want to write custom code to react to an error, such as creating an output report or log dump of the results of the iDiagram's execution, or to the iDiagram being stopped before all entities are finished executing.

The *Error* parameter contains the error number that triggered the event.

Refer to the Entity object documentation for an example that uses all of the events related to entities. See the iGrafx Professional User's Guide for more information about iDiagrams.

Example The following example use the "Main" subroutine to create four shapes that are connected in sequence. An Entity is placed in Shape 1: this is the entity's starting point. To run this example, place the "Main" subroutine in a Diagram-level project, and place the EntitiesAbort event subroutine in a Document-level project (ThisDocument, for example). Run the Main subroutine to create the diagram, then go to the iGrafx Professional interface and Run the entity (using the Entity Manager, or the Run button on the iDiagram toolbar). While the entity is running, press either the Esc key or the Stop button to trigger the EntitiesAbort event.

```
Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnLine As ConnectorLine
    Dim igxEntity As Entity
    ' Create 2 shapes in the diagram and connect them
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
        ixDirWest, ixConnectRelativeToShape)
    ' Create an entity in the first shape
    Set igxEntity = ActiveDocument.Entities.Add("MyEntity", igxShape1)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
        ixDirNorth, ixConnectRelativeToShape)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
```

```
        (1440, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
        ixDirEast, ixConnectRelativeToShape)
    ' Display message box
    MsgBox "Open the Entity Manager dialog and click the Run button." _
        & Chr(13) & "Then press the Stop button or Esc key to abort " _
        & "the Entity."
End Sub
```

Place the following code in a Document-level project, such as ThisDocument.

```
Private Sub AnyShape_EntitiesAbort(ByVal Error As Long)
   Me.AnyShape.FillColor = vbRed
   For iCount = 0 To 3000
       DoEvents
   Next iCount
   MsgBox "The EntityAbort event was triggered."
End Sub
```

See Also EntitiesFinished event

EntitiesStart event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

EntitiesFinished Event

Syntax **Private Sub** *Shape*_**EntitiesFinished**()

Description The EntitiesFinished event occurs when all entities in the document are finished running (complete their execution). You can write this event for the AnyShape control, which means the event fires once for every shape in the document. Using AnyShape is useful for tasks such as collecting the accumulated data in each shape and transferring that data to a database. The event also can be written for individual shapes.

Refer to the Entity object documentation for an example that uses all of the events related to entities. See the iGrafx Professional User's Guide for more information about iDiagrams.

Example The following example use the "Main" subroutine to create four shapes that are connected in sequence. An Entity is placed in Shape 1: this is the entity's starting point. To run this example, place the "Main" subroutine in a Diagram-level project, and place the EntitiesFinished event subroutine in a Document-level project (ThisDocument, for example). Run the Main subroutine to create the diagram, then go to the iGrafx Professional interface and Run the entity (using the Entity Manager, or the Run button on the iDiagram toolbar).

```
Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnLine As ConnectorLine
    Dim igxEntity As Entity
    ' Create 2 shapes in the diagram and connect them
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
        ixDirWest, ixConnectRelativeToShape)
    ' Create an entity in the first shape
    Set igxEntity = ActiveDocument.Entities.Add("MyEntity", igxShape1)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
        ixDirNorth, ixConnectRelativeToShape)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
        ixDirEast, ixConnectRelativeToShape)
    ' Display message box
```

```
    MsgBox "Open the Entity Manager dialog and click the Run button."
End Sub
```

Place the following code in a Document-level project, such as ThisDocument.

```
Private Sub AnyShape_EntitiesFinished()
    Me.AnyShape.FillColor = vbBlack
    For iCount = 0 To 3000
        DoEvents
    Next iCount
    MsgBox "All entities finished. Shape turns black when " _
         & "EntititesFinished event" & Chr(13) & "has completed " _
         & " for that Shape "
End Sub
```

See Also EntitiesAbort event

EntitiesStart event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

EntitiesStart Event

Syntax **Private Sub** *Shape*_**EntitiesStart**()

Description The EntitiesStart event occurs when an iDiagram begins execution; that is, a Run command is issued either through the user interface or programmatically (Document.Run or Entity.Run method). This event is fired for every shape in the document. Custom code can be written within this event procedure to perform any desired actions, such as initializing data structures or beginning an output log.

Refer to the Entity object documentation for an example that uses all of the events related to entities. See the iGrafx Professional User's Guide for more information about iDiagrams.

Example The following example use the "Main" subroutine to create four shapes that are connected in sequence. An Entity is placed in Shape 1: this is the entity's starting point. To run this example, place the "Main" subroutine in a Diagram-level project, and place the EntitiesStart event subroutine in a Document-level project (ThisDocument, for example). Run the Main subroutine to create the diagram, then go to the iGrafx Professional interface and Run the entity (using the Entity Manager, or the Run button on the iDiagram toolbar).

```
Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnLine As ConnectorLine
    Dim igxEntity As Entity
    ' Create 2 shapes in the diagram and connect them
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
        ixDirWest, ixConnectRelativeToShape)
    ' Create an entity in the first shape
    Set igxEntity = ActiveDocument.Entities.Add("MyEntity", igxShape1)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
        ixDirNorth, ixConnectRelativeToShape)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
        ixDirEast, ixConnectRelativeToShape)
    ' Display message box
```

```
    MsgBox "Open the Entity Manager dialog and click the Run button."
End Sub
```

Place the following code in a Document-level project, such as ThisDocument.

```
Private Sub AnyShape_EntitiesStart()
   Me.AnyShape.FillColor = vbGreen
    Me.AnyShape.Text = "Starting"
    For iCount = 0 To 3000
        DoEvents
   Next iCount
    MsgBox "The EntitiesStart event was fired."
End Sub
```

See Also

EntitiesAbort event

EntitiesFinished event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

EntityAccept Event

Syntax **Private Sub** *Shape*_**EntityAccept**(*AcceptEntity* As Boolean, *Entity* As Entity)

Description The EntityAccept event occurs before the entity enters the specified Shape object. Essentially, before an entity enters a shape, the shape is queried, or "asked" whether it will accept the entity. You can use the EntityAccept event to set up criteria for an entity's acceptance into a shape, as well as run other code if the entity is accepted.

The *AcceptEntity* parameter controls whether the entity is accepted or is stopped

The event parameters provide the following data:

- The *AcceptEntity* parameter, when set to False, prevents the entity from entering the shape. The entity stops, and the EntitiesAbort event is triggered. If True, the entity enters the shape and moves on to the EntityExecute event.
- The *Entity* parameter contains the Entity object. This is the entity that is about to enter the shape. You can set custom properties or custom data on the Entity object by using this parameter.

Refer to the Entity object documentation for an example that uses all of the events related to entities. See the iGrafx Professional User's Guide for more information about iDiagrams.

Example The following example use the "Main" subroutine to create four shapes that are connected in sequence. An Entity is placed in Shape 1: this is the entity's starting point. To run this example, place the "Main" subroutine in a Diagram-level project, and place the EntityAccept event subroutine in a Document-level project (ThisDocument, for example). Run the Main subroutine to create the diagram, then go to the iGrafx Professional interface and Run the entity (using the Entity Manager, or the Run button on the iDiagram toolbar).

```
Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnLine As ConnectorLine
    Dim igxEntity As Entity
    ' Create 2 shapes in the diagram and connect them
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
        ixDirWest, ixConnectRelativeToShape)
    ' Create an entity in the first shape
    Set igxEntity = ActiveDocument.Entities.Add("MyEntity", igxShape1)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
        ixDirNorth, ixConnectRelativeToShape)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
```

```
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
        ixDirEast, ixConnectRelativeToShape)
    ' Display message box
    MsgBox "Open the Entity Manager dialog and click the Run button."
End Sub
```

Place the following code in a Document-level project, such as ThisDocument.

```
Private Sub AnyShape_EntityAccept(AcceptEntity As Boolean, ByVal Entity As
IGrafx2.IXEntity)
   Me.AnyShape.FillColor = vbBlue
    Me.AnyShape.Text = "Accepted"
   For iCount = 0 To 3000
       DoEvents
   Next iCount
    MsgBox "The " & Entity.Name & " entity was accepted."
End Sub
```

See Also

EntityExecute event

EntityInitiate event

EntityLeave event

EntityStep event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

EntityExecute Event

Syntax **Private Sub** *Shape*_**EntityExecute**(*Entity* As Entity)

Description The EntityExecute event occurs after the entity has entered the specified Shape object. This results from the entity being "accepted;" that is, the EntityAccept event did not set its AcceptEntity parameter to False.

When an entity executes, it runs the VBA code that is associated with the shape. This is the primary event for iDiagrams, and is where you typically put the code that performs the tasks you want when an entity is in the shape.

The event parameters provide the following data:

- The *Entity* parameter returns an Entity object. This is the entity that is executing in the shape. You can set custom properties or custom data on the Entity by using this parameter.

Refer to the Entity object documentation for an example that uses all of the events related to entities. See the iGrafx Professional User's Guide for more information about iDiagrams.

Note If you want to specifically set the path an entity will travel when leaving the shape, you must set that path in the EntityExecute event.

Example The following example use the "Main" subroutine to create four shapes that are connected in sequence. An Entity is placed in Shape 1: this is the entity's starting point. To run this example, place the "Main" subroutine in a Diagram-level project, and place the EntityExecute event subroutine in a Document-level project (ThisDocument, for example). Run the Main subroutine to create the diagram, then go to the iGrafx Professional interface and Run the entity (using the Entity Manager, or the Run button on the iDiagram toolbar).

```
Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnLine As ConnectorLine
    Dim igxEntity As Entity
    ' Create 2 shapes in the diagram and connect them
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
        ixDirWest, ixConnectRelativeToShape)
    ' Create an entity in the first shape
    Set igxEntity = ActiveDocument.Entities.Add("MyEntity", igxShape1)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
        ixDirNorth, ixConnectRelativeToShape)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
```

```
        (1440, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
        ixDirEast, ixConnectRelativeToShape)
    ' Display message box
    MsgBox "Open the Entity Manager dialog and click the Run button."
End Sub
```

Place the following code in a Document-level project, such as ThisDocument.

```
Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
    Me.AnyShape.FillColor = vbCyan
    Me.AnyShape.Text = "Executing"
    Entity.Size = ixEntityLarge
    For iCount = 0 To 3000
        DoEvents
    Next iCount
    Entity.Size = ixEntityNormal
End Sub
```

See Also

EntityAccept event

EntityInitiate event

EntityLeave event

EntityStep event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

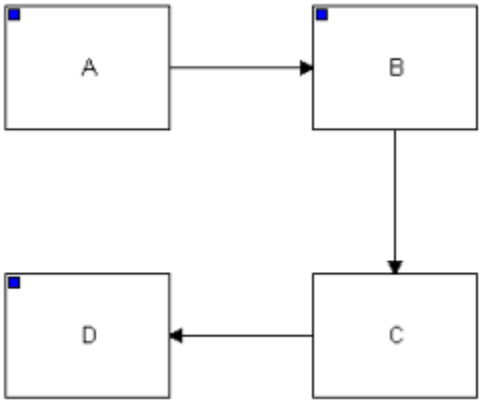
EntityInitiate Event

Syntax **Private Sub** *Shape*_**EntityInitiate**(*Entity* As Entity)

Description The EntityInitiate event occurs once for each shape in the document that contains an entity when Run is selected to start the iDiagram. You can use this event to initialize and prepare each entity before the run of an iDiagram. For example, you might use the EntityInitiate event to clear out custom properties or custom data on an entity.

As an example, in the following diagram there is an entity in Shape A, Shape B, and Shape D. When you press "Run" to start the iDiagram, the EntityInitiate event is triggered for Shapes A, B, and D, but not for Shape C. The value of the *Entity* parameter is the entity that is in Shape A, Shape B, and Shape D, respectively.

Compare this event to the EntitiesStart event, which occurs for every shape in the document after Run is initiated.



The event parameters provide the following data:

- The *Entity* parameter returns an Entity object. This is the entity that is executing in the shape. You can set custom properties or custom data for the Entity object by using this parameter.

Refer to the Entity object documentation for an example that uses all of the events related to entities. See the iGrafx Professional User's Guide for more information about iDiagrams.

Example The following example use the "Main" subroutine to create four shapes that are connected in sequence. An Entity is placed in Shape 1: this is the entity's starting point. To run this example, place the "Main" subroutine in a Diagram-level project, and place the EntityInitiate event subroutine in a Document-level project (ThisDocument, for example). Run the Main subroutine to create the diagram, then go to the iGrafx Professional interface and Run the entity (using the Entity Manager, or the Run button on the iDiagram toolbar).

```
Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnLine As ConnectorLine
    Dim igxEntity As Entity
    ' Create 2 shapes in the diagram and connect them
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
```

```
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
        ixDirWest, ixConnectRelativeToShape)
    ' Create an entity in the first shape
    Set igxEntity = ActiveDocument.Entities.Add("MyEntity", igxShape1)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
        ixDirNorth, ixConnectRelativeToShape)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
        ixDirEast, ixConnectRelativeToShape)
    ' Display message box
    MsgBox "Open the Entity Manager dialog and click the Run button."
End Sub
```

Place the following code in a Document-level project, such as ThisDocument.

```
Private Sub AnyShape_EntityInitiate(ByVal Entity As IGrafx2.IXEntity)
Me.AnyShape.FillColor = vbMagenta
Me.AnyShape.Text = "Initiate"
   For iCount = 0 To 3000
       DoEvents
   Next iCount
   MsgBox "The EntityInitiate event has fired."
End Sub
```

See Also

EntityAccept event

EntityExecute event

EntityLeave event

EntityStep event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

EntityLeave Event

Syntax **Private Sub** *Shape*_**EntityLeave**(*Entity* As Entity)

Description The EntityLeave event occurs when the entity is leaving the specified Shape object. This event happens after the EntityExecute event and before the EntityStep event.

The event parameters provide the following data:

- The *Entity* parameter returns an Entity object. This is the entity that is executing in the shape. You can set custom properties or custom data on the Entity by using this parameter

Refer to the Entity object documentation for an example that uses all of the events related to entities. See the iGrafx Professional User's Guide for more information about iDiagrams.

Example The following example use the "Main" subroutine to create four shapes that are connected in sequence. An Entity is placed in Shape 1: this is the entity's starting point. To run this example, place the "Main" subroutine in a Diagram-level project, and place the EntityLeave event subroutine in a Document-level project (ThisDocument, for example). Run the Main subroutine to create the diagram, then go to the iGrafx Professional interface and Run the entity (using the Entity Manager, or the Run button on the iDiagram toolbar).

```
Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnLine As ConnectorLine
    Dim igxEntity As Entity
    ' Create 2 shapes in the diagram and connect them
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
        ixDirWest, ixConnectRelativeToShape)
    ' Create an entity in the first shape
    Set igxEntity = ActiveDocument.Entities.Add("MyEntity", igxShape1)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
        ixDirNorth, ixConnectRelativeToShape)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
```

```
        ixDirEast, ixConnectRelativeToShape)
    ' Display message box
    MsgBox "Open the Entity Manager dialog and click the Run button."
End Sub
```

Place the following code in a Document-level project, such as ThisDocument.

```
Private Sub AnyShape_EntityLeave(ByVal Entity As IGrafx2.IXEntity)
    Me.AnyShape.FillColor = vbWhite
    Me.AnyShape.Text = "Leaving"
    For iCount = 0 To 3000
        DoEvents
    Next iCount
    MsgBox "EntityLeave event done for " & AnyShape.ObjectName
End Sub
```

See Also

EntityAccept event

EntityExecute event

EntityInitiate event

EntityStep event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

EntityStep Event

Syntax **Private Sub** *Shape*_**EntityStep**(*Entity* As Entity)

Description The EntityStep event occurs for an entity each time the entity moves. The event is similar to a timer. It occurs after the EntityLeave event, and before the next EntityAccept event.

Code can be written within this event procedure to perform any desired actions. This event can be used to update a graphical representation or accumulate values.

The event parameters provide the following data:

- The *Entity* parameter returns an Entity object. This is the entity that is executing in the shape. You can set custom properties or custom data on the Entity by using this parameter.

Refer to the Entity object documentation for an example that uses all of the events related to entities. See the iGrafx Professional User's Guide for more information about iDiagrams.

Example The following example use the "Main" subroutine to create four shapes that are connected in sequence. An Entity is placed in Shape 1: this is the entity's starting point. To run this example, place the "Main" subroutine in a Diagram-level project, and place the EntityStep event subroutine in a Document-level project (ThisDocument, for example). Run the Main subroutine to create the diagram, then go to the iGrafx Professional interface and Run the entity (using the Entity Manager, or the Run button on the iDiagram toolbar).

```
Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnLine As ConnectorLine
    Dim igxEntity As Entity
    ' Create 2 shapes in the diagram and connect them
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
        ixDirWest, ixConnectRelativeToShape)
    ' Create an entity in the first shape
    Set igxEntity = ActiveDocument.Entities.Add("MyEntity", igxShape1)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
        ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
        ixDirNorth, ixConnectRelativeToShape)
    ' Add a third shape and connect it to shape 2
    Set igxShape1 = igxShape2
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440, 1440 * 3)
    ' Add connector line
    Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
```

```
        ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
        ixDirEast, ixConnectRelativeToShape)
    ' Display message box
    MsgBox "Open the Entity Manager dialog and click the Run button."
End Sub
```

Place the following code in a Document-level project, such as ThisDocument.

```
Private Sub AnyShape_EntityStep(ByVal Entity As IGrafx2.IXEntity)
    Me.AnyShape.FillColor = vbYellow
    Me.AnyShape.Text = "Step event is active"
    For iCount = 0 To 3000
        DoEvents
    Next iCount
    Me.AnyShape.Text = ""
End Sub
```

See Also

EntityAccept event

EntityExecute event

EntityInitiate event

EntityLeave event

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

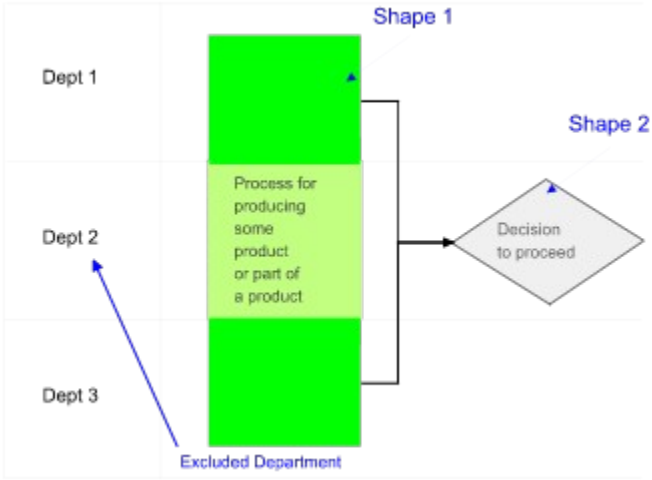
ExcludedDepartmentNames Property

Syntax *Shape*.**ExcludedDepartmentNames**

Data Type ExcludedDepartmentNames collection object (read-only, See Object Properties)

Description The ExcludedDepartmentNames property returns the ExcludedDepartmentNames collection for the specified Shape object.

In the following illustration, the shape has one item in the ExcludedDepartmentNames collection, "Dept. 2".



See the iGrafx Professional User's Guide for more information on excluded departments.

Example The following example creates a shape, then adds a department name to the ExcludedDepartmentNames object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDepartment1 As Department
Dim igxDepartment2 As Department
Dim igxDepartment3 As Department
' Create three departments
Set igxDepartment1 = ActiveDiagram.Departments.Item(1)
Set igxDepartment2 = _
   ActiveDiagram.Departments.AddDepartment("TestDept2")
Set igxDepartment3 = _
   ActiveDiagram.Departments.AddDepartment("TestDept3")
' Create a new shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the top and bottom departments for the shape
igxShape.TopDepartment = igxDepartment1
igxShape.BottomDepartment = igxDepartment3
igxShape.ExcludedDepartmentNames.Add "TestDept2"
MsgBox "Shape created that spans departments."
```

See Also ExcludedDepartmentNames object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

FillColor Property

Syntax *Shape*.**FillColor**

Data Type Color (read/write)

Description The FillColor property defines the foreground fill color for the specified Shape object. It also controls the color of the lines that make up a fill pattern (the background behind a fill pattern is controlled by the BackColor property). Color values are specified with the RGB function, or with one of the Visual Basic color constants.

The value of the FillType property controls how the FillColor property is used.

- If the FillType property is set to IxFillNone, the FillColor property has no effect.
- If the FillType property is set to IxFillSolid, the property specifies the interior fill color. Border lines are not affected by this property.
- If the FillType property is set to IxFillPattern, the FillColor sets the color of the lines that make up the fill pattern. The color of the background behind the pattern lines is controlled by the BackColor property. Refer to the Format—Fill dialog in the iGrafx Professional user interface, or to the iGrafx Professional User's Guide for more information about fill patterns.
- If the FillType property is set to IxFillGradient, the FillColor is used as the StartColor in the gradient style (the BackColor is used as the EndColor). Refer to the Format—Fill dialog in the iGrafx Professional user interface, or to the iGrafx Professional User's Guide for more information about gradients.

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The fill properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a fill property most recently is the one that is used. However, fill properties specified at lower levels, such as the Graphic object do not have precedence. The only exception is that you can use the Graphic object's ProtectFillFormat property to force an override of values set at the Shape level.

For more information about fills, refer to the iGrafx Professional User's Guide, or the Format—Fills dialog.

Example The following example creates a shape on the active diagram, then set the FillColor property for that shape to blue.

```
' Dimension the variables
Dim igxShape As Shape
' Create a new Shape object in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the FillColor of the shape to blue
MsgBox "Click OK to change the fill color to blue."
igxShape.FillColor = vbBlue
MsgBox "Click OK to continue."
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

FillType Property

Syntax *Shape*.**FillType**

Data Type IxFillType enumerated constant (read/write)

Description The FillType property defines the type of fill to use for the specified Shape object. The FillType property can affect other properties; these effects are described in the table below. For information about using fills with graphics in iGrafx Professional, refer to the iGrafx Professional User's Guide.

The IxFillType constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant	Effect on Other Properties
1	ixFillNone	FillColor, GradientIndex, and PatternIndex properties are ignored. BackColor is used to fill in gaps of a line if a broken LineStyle (dashed, dotted, etc) is chosen.
2	ixFillSolid	FillColor sets the foreground interior fill. BackColor is used only if the LineStyle is a broken line. GradientIndex and PatternIndex have no effect.
4	ixFillPattern	FillColor controls the color of the lines that make up the fill pattern. BackColor sets the color behind the pattern lines. PatternIndex sets the pattern to use as the fill. GradientIndex has no effect.
5	ixFillGradient	FillColor is the StartColor of the gradient. BackColor is the EndColor of the gradient. GradientIndex sets the gradient style (type) to use as the fill. PatternIndex has no effect.

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The fill properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a fill property most recently is the one that is used. However, fill properties specified at lower levels, such as the Graphic object do not have precedence. The only exception is that you can use the Graphic object's ProtectFillFormat property to force an override of values set at the Shape level.

For more information about fills, refer to the iGrafx Professional User's Guide, or the Format—Fills dialog.

Error An Index Out of Range error is generated if any value is supplied that is not one of the IxFillType constant values defined in the table above.

Example The following example creates a shape on the active diagram, and then sets the fill type to gradient. The gradient color is then set to blue for the fill color and green for the back color.

```
' Dimension the variables
Dim igxShape As Shape
' Create a new Shape object on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the FillType Property of the shape that was added
igxShape.FillType = ixFillGradient
' Set the GradientIndex Property of the shape that was added
igxShape.GradientIndex = 5
```

```
' Set the FillColor of the shape to blue
igxShape.FillColor = vbBlue
' Set the BackColor of the shape to green
igxShape.BackColor = vbGreen
MsgBox "Click OK to continue"
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

FitToText Method

Syntax *Shape*.**FitToText** ([*FitType* As IxFitToTextType = ixAutoFit])

Description The FitToText method changes the size of the specified Shape object to fit the text contained within it (the shape's Text property).The *FitType* argument specifies how to change the shape to fit the text.

The IxFitToTextType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant	Description
-1	ixAutoFit	Resizes the shape using iGrafx Professional's automatic resizing algorithm.
0	ixPreserveWidth	Resizes the shape by changing the height of the shape, and preserving the current width of the shape.
1	ixPreserveHeight	Resizes the shape by changing the width of the shape, and preserving the current height of the shape.
2	ixPreserveAspectRatio	Resizes the shape by changing both the width and the height to maintain the curent aspect ratio of the shape.
3	ixNoLineWrapping	Resizes the shape so that the text is not wrapped. Only a new paragraph causes a new line of text.

Example The following example creates a shape containing text in the active diagram. It then invokes the FitToText method so that all of the text is contained within the with no line wrapping.

```
' Dimension the variables
Dim igxShape As Shape
' Create a new Shape object on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add text to the shape
igxShape.Text = _
   "Testing the FitToText Property for the shape just added!"
' Shrink the shape so the text doesn't fit
igxShape.DiagramObject.Width = 500
igxShape.DiagramObject.Height = 500
MsgBox "Shape created. Click OK to FitToText."
' Make the shape fit the text while preserving the aspect ratio
igxShape.FitToText ixNoLineWrapping
MsgBox "Click OK to continue"
```

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

GradientIndex Property

Syntax *Shape*.**GradientIndex**

Data Type Integer (read/write)

Description The GradientIndex property specifies the gradient type (or pattern) to use as the fill for the specified Shape object. This property is valid only if the shape's FillType property is set to a value of ixFillGradient.

The value can be any integer value between zero and the number of gradients available. Since you can create new gradients, the number of gradients may vary from one installation of iGrafx Professional to another.

The FillColor property is used as the start color of the gradient, and the BackColor property is used as the end color of the gradient.

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The fill properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a fill property most recently is the one that is used. However, fill properties specified at lower levels, such as the Graphic object do not have precedence.

The Graphic object also has a GradientIndex property. If the property is set at both the Shape and Graphic level, the shape's property is used unless the Graphic object's ProtectFillFormat property is set to True, which causes the Graphic object's GradientIndex property to override the shape's GradientIndex property.

For more information about gradients, refer to the iGrafx Professional User's Guide, or the Format—Fills dialog.

Error An Index Out of Range error is generated if the index is less than zero or is larger than the last valid index value.

Example The following example creates a shape on the active diagram, and then sets the fill type to gradient. The gradient color is then set to blue for the fill color and green for the back color.

```
' Dimension the variables
Dim igxShape As Shape
' Create a new Shape object on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the FillType Property of the shape that was added
igxShape.FillType = ixFillGradient
' Set the GradientIndex Property of the shape that was added
igxShape.GradientIndex = 5
' Set the FillColor of the shape to blue
igxShape.FillColor = vbBlue
' Set the BackColor of the shape to green
igxShape.BackColor = vbGreen
MsgBox "Click OK to continue"
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

Graphic Property

Syntax *Shape*.**Graphic**

Data Type Graphic object (read-only, See Object Properties)

Description The Graphic property returns the Graphic object for the specified Shape object. The Graphic property can be used to manipulate the visible portion of a shape. See the Graphic object for more information.

Example The following example illustrates how to use the Replace method to replace the graphic of an existing shape with one created in a GraphicBuilder object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim igxGraphicBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a rectangle to the graphic
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Set the color of the rectangle to red
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
' Add an ellipse to the graphic
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the color of the ellipse to blue
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
   FillFormat.FillColor = vbBlue
' Add a 5 point star to the graphic
igxGraphicBuilder.Star 0.5, 0.5, 0.3, 0.15, 5, 30
' Set the color of the star to green
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(3). _
   FillFormat.FillColor = vbGreen
' Replace the graphic inside the shape with the new graphic
MsgBox "Click OK to replace the graphic."
igxShape.Graphic.Replace igxGraphicBuilder.Graphic
MsgBox "Click OK to continue."
```

See Also

Graphic object

GraphicGroup object

GraphicBuilder object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

InputConnectorLines Property

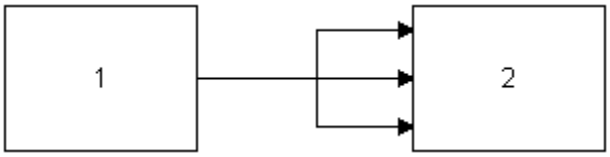
Syntax *Shape*.**InputConnectorLines**

Data Type ObjectRange object (read-only, See Object Properties)

Description The InputConnectorLines property returns an ObjectRange collection that contains all the connector lines that have the specified Shape object as a destination.

InputConnectorLines compared to InputPaths

The number of input connector lines can differ from the number of input paths. This can occur because a single path can be made up of multiple connector lines. For example in the illustration below, there is one connector line connected to Shape 1.



The other branches connect into the line and not Shape 1. So, Shape 2 would have one input connector line, but three input paths. Shape 1 would have one output connector line and three output paths.

The arrowheads on a line do not necessarily indicate whether a connector line is an input or an output to a shape. A connector line is an input connector line to a shape if the connector line was originally drawn to the shape from another shape.

Example The following example creates three new shapes and connects them with connector lines. The InputConnectorLines property is used to access any connector lines that are inputs for Shape 2. Any connector lines in that collection are highlighted by changing their color.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxObjects As DiagramObjects
' Get the DiagramObjects object
Set igxObjects = ActiveDiagram.DiagramObjects
' Create 3 new shapes and set their labels
Set igxShape1 = igxObjects.AddShape(1440, 1440)
Set igxShape2 = igxObjects.AddShape(1440 * 3, 1440 * 2)
Set igxShape3 = igxObjects.AddShape(1440 * 5, 1440 * 2)
igxShape1.Text = "Shape 1"
igxShape2.Text = "Shape 2"
igxShape3.Text = "Shape 3"
' Add connector lines between the new shapes
Set igxConnector1 = igxObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
Set igxConnector2 = igxObjects.AddConnectorLine _
    (, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
```

```
' Change the color of connectors if they are inputs for Shape 2
MsgBox "Click OK to change the color of any " _
    & "connector lines which input Shape 2."
igxShape2.InputConnectorLines.LineFormat.Color = vbBlue
MsgBox "Click OK to continue."
```

See Also

OutputConnectorLines property

OutputPaths property

ObjectRange object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

IsCrossDepartment Property

Syntax *Shape*.**IsCrossDepartment**[= {**True | False**}]

Data Type Boolean (read-only)

Description The IsCrossDepartment property indicates whether a shape occupies more than one Department in the diagram. If the shape crosses more than one department, then the IsCrossDepartment property is "True". If the shape occupies only one department, then the IsCrossDepartment property is "False".

Example The following example sets up several departments, and one shape that spans multiple departments. A message indicates if the shape crosses departments, based on the IsCrossDepartment property.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDepartment1 As Department
Dim igxDepartment2 As Department
Dim igxDepartment3 As Department
' Create three departments
Set igxDepartment1 = _
   ActiveDiagram.Departments.AddDepartment("TestDept1")
Set igxDepartment2 = _
   ActiveDiagram.Departments.AddDepartment("TestDept2")
Set igxDepartment3 = _
   ActiveDiagram.Departments.AddDepartment("TestDept3")
' Create a new shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "View the diagram"
' Set the top and bottom departments for the shape
igxShape.TopDepartment = igxDepartment1
igxShape.BottomDepartment = igxDepartment3
igxShape.ExcludedDepartmentNames.Add "TestDept2"
If igxShape.IsCrossDepartment Then
   MsgBox "This shape crosses more than one department."
Else
   MsgBox "This shape occupies only one department."
End If
```

See Also DepartmentRange property

ExcludedDepartmentNames property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

IsDecision Property

Syntax *Shape*.**IsDecision**[= {**True | False**}]

Data Type Boolean (read-only)

Description The IsDecision property indicates whether the specified shape is a decision shape; that is, it has decision cases. If the property returns True, it means that the shape has a valid DecisionCase object. A value of False means that the shape does not have a valid DecisionCase object.

Example The following example creates a simple decision diagram with four shapes. The second shape is a decision point, and to it is added two decision cases, a Yes and a No branch. Then the IsDecision property is used to determine which of the shapes has decision cases in order to make its fill color yellow rather than green.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine As ConnectorLine
Dim igxDiagObj As DiagramObject
' Create shapes in the diagram for a decision structure
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries(1)(1))
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2, Application.ShapeLibraries(1)(3))
' Label the shapes
igxShape1.Text = "Start"
igxShape2.Text = "Decision"
' Connect shapes 1 and 2
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Add decision cases to shape 2
Call igxShape2.DecisionCases.Add("Yes", 0.6)
Call igxShape2.DecisionCases.Add("No", 0.4)
Set igxShape1 = igxShape2
' Add next shape and label it
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 2, Application.ShapeLibraries(1)(1))
igxShape2.Text = "No path"
' Connect shapes 2 and 3
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Add next shape and label it
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 4, Application.ShapeLibraries(1)(1))
igxShape2.Text = "Yes path"
' Connect shapes 2 and 4
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
    ixDirNorth, ixConnectRelativeToShape)
MsgBox "View the diagram"
```

```
' Find the Decision shape. Make its fill yellow, and all other
' shapes fill with green
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
    Set igxDiagObj = ActiveDiagram.DiagramObjects.Item(iCount)
    If (igxDiagObj.Type = ixObjectShape) Then
        If (igxDiagObj.Shape.IsDecision) Then
            igxDiagObj.Shape.FillColor = vbYellow
        Else
            igxDiagObj.Shape.FillColor = vbGreen
        End If
    End If
    MsgBox "View the diagram"
Next iCount
```

See Also

DecisionCases property

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

IsStartPoint Property

Syntax *Shape*.**IsStartPoint**[= {**True** | **False**}]

Data Type Boolean (read/write)

Description The IsStartPoint property indicates whether the specified Shape object is the starting point for a sub-process. Starting points can be linked to from shapes in the same diagram or from other diagrams within a document, allowing iDiagram Entities to move between diagrams in a document.

The property's value is True if a name has been assigned to the StartPointName property. The property's value is False if a name has not been assigned to the StartPointName property. Furthermore, you cannot use this property to change a shape into a start point merely by changing this property from False to True. The only way to make a shape a start point is to assign a name the StartPointName property. However, you can do the reverse: you can remove a start point designation from a shape by changing this property from True to False. Refer to the StartPointName property for additional information.

Example The following example creates three shapes in a diagram, and makes the third shape a start point by assigning a name to the shape's StartPointName property. It then queries the IsStartPoint property, and displays a message indicating which shapes are start points. The final part of the code shows that the IsStartPoint property cannot be used to create a start point, but can be used to remove a start point.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDiagramObjects As DiagramObjects
Dim igxDiagramObject As DiagramObject
' Get DiagramObjects collection from active diagram
Set igxDiagramObjects = ActiveDiagram.DiagramObjects
' Create several shapes in the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 2)
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440)
MsgBox "Shapes created. Click OK to add a StartPointName."
' Set the StartPointName for the third shape
igxShape.StartPointName = "StartPoint1"
' Find the shapes that are start points
For Each igxDiagramObject In igxDiagramObjects
    igxDiagramObject.Selected = True
    ' If shape is a starting point, display a message
    If igxDiagramObject.Shape.IsStartPoint Then
        MsgBox "This shape is a start point. Links can jump to it." _
            & Chr(13) & "The start point name is " _
            & igxShape.StartPointName
    Else
        MsgBox "This shape is not a start point"
    End If
    igxDiagramObject.Selected = False
Next igxDiagramObject
' Try to make the second shape a start point
For iCount = 1 To igxDiagramObjects.Count
    If iCount = 2 Then
        igxDiagramObjects.Item(iCount).Shape.IsStartPoint = True
```

```
    End If
Next iCount
' Find the shapes that are start points
For Each igxDiagramObject In igxDiagramObjects
   igxDiagramObject.Selected = True
   ' If shape is a starting point, display a message
   If igxDiagramObject.Shape.IsStartPoint Then
       MsgBox "This shape is a start point. Links can jump to it." _
           & Chr(13) & "The start point name is " _
           & igxShape.StartPointName
   Else
       MsgBox "This shape is not a start point"
   End If
   igxDiagramObject.Selected = False
Next igxDiagramObject
' Remove the start point assignment from the third shape
For iCount = 1 To igxDiagramObjects.Count
    If iCount = 3 Then
        igxDiagramObjects.Item(iCount).Shape.IsStartPoint = False
    End If
Next iCount
' Find the shapes that are start points
For Each igxDiagramObject In igxDiagramObjects
   igxDiagramObject.Selected = True
   ' If shape is a starting point, display a message
   If igxDiagramObject.Shape.IsStartPoint Then
       MsgBox "This shape is a start point. Links can jump to it." _
           & Chr(13) & "The start point name is " _
           & igxShape.StartPointName
   Else
       MsgBox "This shape is not a start point"
   End If
   igxDiagramObject.Selected = False
Next igxDiagramObject
```

See Also StartPointName property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

LineColor Property

Syntax *Shape*.**LineColor**

Data Type Color (read/write)

Description The LineColor property specifies the color for the line used to draw the border of a Shape object. The property is ignored if the shape's LineStyle property is set to ixLineNone. You can specify the color using any method that is valid in Visual Basic programming (refer to your Visual Basic programming documentation).

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The line properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a line property most recently is the one that is used. However, line properties specified at lower levels, such as the Graphic object do not have precedence. The only exception is that you can use the Graphic object's ProtectLineFormat property to force an override of values set at the Shape level.

For more information about lines and borders properties, refer to the iGrafx Professional User's Guide, or the Format—Line and Borders dialog.

Example The following example creates a shape, then sets the LineColor, LineStyle, and LineWidth properties for the shape's border. It then sets these same properties through the ShapeFormat object, showing that the value set most recently is used.

```
' Dimension the variables
Dim igxShape As Shape
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the LineColor property for the shape's border
igxShape.LineColor = vbGreen
' Set the LineStyle property for the shape's border
igxShape.LineStyle = ixLineDashed
' Set the LineWidth property for the shape's border
igxShape.LineWidth = 100
MsgBox "View the diagram"
Set igxShapeFmt = igxShape.ShapeFormat
With igxShapeFmt.LineFormat
    .Color = vbRed
    .Style = ixLineNormal
    .Width = 40
End With
MsgBox "View the diagram"
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

LineStyle Property

Syntax *Shape*.**LineStyle**

Data Type IxLineStyle enumerated constant (read/write)

Description The LineStyle property specifies the style of the line used to draw the border of the specified Shape object. Line styles are solid, dashed, etc.

The LineColor property controls the color of the lines. If the line style is set to one of the broken line types (dashed, dotted, etc.), then the BackColor property colors the spaces between the dashes, dots, etc. The width of the line is controlled by the LineWidth property.

The IxLineStyle constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
-2	ixLineNone
0	ixLineNormal
1	ixLineDashed
2	ixLineDotted
3	ixLineDashDot
4	ixLineDashDotDot

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The line properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a line property most recently is the one that is used. However, line properties specified at lower levels, such as the Graphic object do not have precedence. The only exception is that you can use the Graphic object's ProtectLineFormat property to force an override of values set at the Shape level.

For more information about lines and borders properties, refer to the iGrafx Professional User's Guide, or the Format—Line and Borders dialog.

Example The following example creates a shape, then sets the LineColor, LineStyle, and LineWidth properties for the shape's border. It then sets these same properties through the ShapeFormat object, showing that the value set most recently is used.

```
' Dimension the variables
Dim igxShape As Shape
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the LineColor Property for the shape's border
igxShape.LineColor = vbGreen
' Set the LineStyle Property for the shape's border
igxShape.LineStyle = ixLineDashed
' Set the LineWidth Property for the shape's border
igxShape.LineWidth = 100
MsgBox "View the diagram"
Set igxShapeFmt = igxShape.ShapeFormat
With igxShapeFmt.LineFormat
    .Color = vbRed
    .Style = ixLineNormal
    .Width = 40
End With
```

```
MsgBox "View the diagram"
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

LineWidth Property

Syntax *Shape*.**LineWidth**

Data Type Integer (read/write)

Description The LineWidth property specifies the width of the line used to draw the border of a Shape object. This property is ignored if the LineStyle property is set to ixLineNone.

Valid values for this property are specified in Twips, and can be between 0 and 100. This contrasts with the user interface, where line width values are specified in points (1 point = 1/72 inch = 20 twips). A value of zero creates a very fine hairline. A value of 20 creates a one point line, 40 a two point line, 60 a three point line, etc.

The user interface rounds off the displayed point value to the nearest whole point, but the original twips value remains intact, and the line is drawn and printed using the precise twips value. The following tables show the relationship between values set in Visual Basic, and values set from the user interface:

Set the Visual Basic LineWidth property to:	**Line Width Displayed in the Format Shape Dialog Box in the User Interface**
0 – 9 twips	Hairline
10 – 29 twips	1 point
30 – 49 twips	2 point
50 – 69 twips	3 point
70 – 89 twips	4 point
90 – 100 twips	5 point

Set the User Interface Line Width to:	**Line Width value in Visual Basic:**
Hairline	0 twips
1 point	20 twips
2 point	40 twips
3 point	60 twips
4 point	80 twips
5 point	100 twips

Allowing the programmer to specify the line width in twips provides for finer control of the line; for instance, you could specify a 2.5 point line (50 twips) or a 2.25 point line (45 twips). The point value represented in the user interface is rounded; for instance, a 50 twip line rounds to 3 in the user interface, and a 49 twip line rounds to 2 in the user interface. This rounding does not affect the actual value you set. However, be aware that a user can change a value that you set by using the Lines and Borders dialog.

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The line properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a line property most recently is the one that is used. However, line properties specified at lower levels, such as the Graphic object do not have precedence. The only exception is that you can use the Graphic object's ProtectLineFormat property to force an override of values set at the Shape level.

For more information about lines and borders properties, refer to the iGrafx Professional User's Guide, or the Format—Line and Borders dialog.

Example The following example creates a shape, then sets the LineColor, LineStyle, and LineWidth properties for the shape's border.

```
' Dimension the variables
Dim igxShape As Shape
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the LineColor Property for the shape's border
igxShape.LineColor = vbGreen
' Set the LineStyle Property for the shape's border
igxShape.LineStyle = ixLineDashed
' Set the LineWidth Property for the shape's border
igxShape.LineWidth = 100
MsgBox "View the diagram"
Set igxShapeFmt = igxShape.ShapeFormat
With igxShapeFmt.LineFormat
    .Color = vbRed
    .Style = ixLineNormal
    .Width = 40
End With
MsgBox "View the diagram"
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

Links Property

Syntax *Shape*.**Links**

Data Type Links collection object (read-only, See Object Properties)

Description The Links property returns the Links collection for the specified Shape object. The Links collection allows you to add, delete, or modify the links of the shape.

Example The following example creates a shape on the active diagram, and then creates a link for the shape by adding a link to the Links collection.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxLinks As Links
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the Links collection from the shape
Set igxLinks = igxShape.Links
' Add a new link to the shape
MsgBox "Shape created. Click OK to add a link to the shape."
igxLinks.AddDiagramLink ("Diagram2")
MsgBox "The link is called " & igxLinks.Item(1)
```

See Also Link object

Links object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

Note Property

Syntax *Shape*.**Note**

Data Type Note object (read-only, See Object Properties)

Description The Note property returns the Note object for the specified Shape object. The Note object can be used to add comments about a shape that user's can view, add to, or remove.

Example The following example creates a shape on the active diagram, and then sets the text of the note with the date and time the shape was created.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxNote As Note
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the Note object from the shape
Set igxNote = igxShape.Note
' Set the note with the date and time the object was created
igxNote.Text = "This is some text for the note of this shape." _
    & Chr(13) & "It was created at: " & Now
MsgBox "The following is the shape's Note: " & Chr(13) _
    & Chr(13) & igxShape.Note.Text
```

See Also Note object

iGrafx API Object Hierarchy

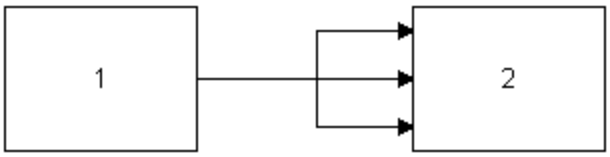
{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

OutputConnectorLines Property

Syntax *Shape*.**OutputConnectorLines**

Data Type ObjectRange object (read-only, See Object Properties)

Description The OutputConnectorLines property returns an ObjectRange object that contains the connector lines that are outputs from the specified Shape object. The number of output connector lines can differ from the number of output paths. This can occur because a single path can be made up of multiple connector lines. For example, in the following illustration there is one connector line connected to Shape 1.



The other branches connect into the line and not Shape 1. So, Shape 2 would have one input connector line, but three input paths. Shape 1 would have one output connector line and three output paths.

The arrowheads on a line do not necessarily indicate whether a connector line is input or output to a shape. A connector line is an output *from* a shape if the connector line was originally drawn *from* the shape to another shape.

Example The following example creates three shapes connected with connector lines. Using the OutputConnectorLines property, any connector lines which are outputs of Shape 2 are highlighted, by changing the color to blue.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxObjects As DiagramObjects
' Get the DiagramObjects object
Set igxObjects = ActiveDiagram.DiagramObjects
' Create 3 new shapes and set their labels
Set igxShape1 = igxObjects.AddShape(1440, 1440)
Set igxShape2 = igxObjects.AddShape(1440 * 3, 1440)
Set igxShape3 = igxObjects.AddShape(1440 * 5, 1440)
igxShape1.Text = "Shape 1"
igxShape2.Text = "Shape 2"
igxShape3.Text = "Shape 3"
' Add connector lines between the new shapes
Set igxConnector1 = igxObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
Set igxConnector2 = igxObjects.AddConnectorLine _
    (, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
' Change the color of connectors if they are inputs for Shape 2
MsgBox "Click OK to change the color of any connector lines which are outputs
of Shape 2."
```

```
igxShape2.OutputConnectorLines.LineFormat.Color = vbBlue
MsgBox "Click OK to continue."
```

See Also

InputConnectorLines property

OutputPaths property

ObjectRange object

iGrafx API Object Hierarchy

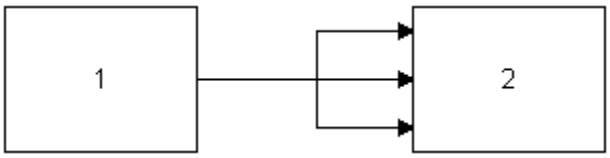
{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

OutputPaths Property

Syntax *Shape*.**OutputPaths**

Data Type Paths collection object (read-only, See Object Properties)

Description The OutputPaths property returns a Paths collection that contains the output paths for the specified Shape object. The number of output connector lines can differ from the number of output paths. This can occur because a single path can be made up of multiple connector lines. For example, in the following illustration there is one connector line connected to Shape 1.



The other branches connect into the line and not Shape 1. So, Shape 2 would have one input connector line, but three input paths. Shape 1 would have one output connector line and three output paths.

The arrowheads on a line do not necessarily indicate whether a connector line is input or output to a shape. A connector line is an output *from* a shape if the connector line was originally drawn *from* the shape to another shape.

Note: This property replaces the AvailablePaths property of the previous API's.

Example The following example creates three shapes connected with connector lines. It then displays the number of OutputPaths for each of the shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxObjects As DiagramObjects
' Get the DiagramObjects object
Set igxObjects = ActiveDiagram.DiagramObjects
' Create 3 new shapes and set their labels
Set igxShape1 = igxObjects.AddShape(1440, 1440)
Set igxShape2 = igxObjects.AddShape(1440 * 3, 1440)
Set igxShape3 = igxObjects.AddShape(1440 * 5, 1440)
igxShape1.Text = "Shape 1"
igxShape2.Text = "Shape 2"
igxShape3.Text = "Shape 3"
' Add connector lines between the new shapes
Set igxConnector1 = igxObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
Set igxConnector2 = igxObjects.AddConnectorLine _
    (, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
' Change the color of connectors if they are inputs for Shape 2
MsgBox "Shape 1 has " & igxShape1.OutputPaths.Count _
    & " OutputPaths." & Chr(13) & "Shape 2 has " _
```

```
& igxShape2.OutputPaths.Count & " OutputPaths." & _
Chr(13) & "Shape 3 has " & igxShape3.OutputPaths.Count _
& " OutputPaths."
```

See Also

OutputConnectorLines property

InputConnectorLines property

Path object

Paths object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

PatternIndex Property

Syntax *Shape*.**PatternIndex**

Data Type Integer (read/write)

Description The PatternIndex property specifies the fill pattern to use as the fill for the specified Shape object. Valid values range from 0 to 32. This property is valid only if the FillType property is set to a value of ixFillPattern.

When using pattern fills, the FillColor property defines the color of the lines that make up the pattern, and the BackColor property defines the color behind the pattern of lines.

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The fill properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a fill property most recently is the one that is used. However, fill properties specified at lower levels, such as the Graphic object do not have precedence.

The Graphic object also has a PatternIndex property. If the property is set at both the Shape and Graphic level, the shape's property is used unless the Graphic object's ProtectFillFormat property is set to True, which causes the Graphic object's PatternIndex property to override the shape's PatternIndex property.

For more information about pattern fills, refer to the iGrafx Professional User's Guide, or the Format—Fills dialog.

Errors An index out of range error is returned if the index value supplied is less than 0 or greater than 32.

Example The following example creates a shape on the active diagram, sets the fill type to pattern with a blue foreground color and green background color. It then sets five different pattern fills on the shape, selected at random.

```
' Dimension the variables
Dim igxShape As Shape
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the FillType property to pattern
igxShape.FillType = ixFillPattern
' Set the FillColor property to blue
igxShape.FillColor = vbBlue
' Set the BackColor property to green
igxShape.BackColor = vbGreen
' Show 5 randomly chosen pattern fills
For iLoop = 1 To 5
    MsgBox "Click OK for a randomly selected pattern fill"
    igxShape.PatternIndex = Rnd(1) * 32
Next iLoop
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

PermanentShape Property

Syntax *Shape*.**PermanentShape**

Data Type Shape object (read-only, See Object Properties)

Description The PermanentShape property returns a Shape object. The purpose of this property is to provide a means of holding on to the object an AnyControl is pointing at after an event is over.

The AnyControl objects are special VBA controls that are only valid during an event; these objects dynamically point at the "active" object that is triggering the event. The PermanentShape property is used to "grab" the specific object the AnyControl is pointing at so that it can be used (or accessed) once the event is over.

As an example, consider the following event procedure written for the AnyShape_Select event.

```
Private Sub AnyShape_Select()
    Set MyShape = AnyShape
End Sub
```

If the variable MyShape is a global variable of type Shape, then within the Select event you can set MyShape to the Shape object that is currently active. However, if you try to use MyShape after the event is over, it returns an error because an event is not in progress. Since you set MyShape to the AnyControl, your variable is pointing at the AnyControl that is dynamically pointing at the active object, which is Nothing outside of an event.

If your intent is to hold on to the specific shape that the AnyShape control is pointing at inside the event, then you need to use the PermanentShape property. This property gives you a Shape object that is valid after the event is over (outside of the event). The change to your code is as follows (MyShape is a global variable of type Shape):

```
Private Sub AnyShape_Select()
    Set MyShape = AnyShape.PermanentShape
End Sub
```

Example The following example defines two subroutines and an event. The first subroutine "MakeShapes ( )" puts two shapes in the diagram. Drag a connector line between the shapes after running this subroutine. This triggers the AfterConnectorAttach event, and sets the permanent shape. Next, run the second subroutine, which affects the permanent shape that was captured during the event.

```
Public igxShape As Shape

Public Sub MakeShapes()  'Run first
   ' Add two shapes to the diagram
   ActiveDiagram.DiagramObjects.AddShape 1440, 1440
   ActiveDiagram.DiagramObjects.AddShape 1440 * 4, 1440
   ' Display a message that the event is ready
   MsgBox "Shapes created. The event is now active." _
       & Chr(13) & _
       "Return to the diagram and try dragging a connector" _
       & Chr(13) & _
       "line between the shapes. Then run the next subroutine."
End Sub

Public Sub ChangePermanentShape() 'Run second
   MsgBox "Click OK to change the permanent shape to white."
```

```
   igxShape.FillColor = vbWhite
   MsgBox "Click OK to continue."
End Sub

Private Sub AnyShape_AfterConnectorAttach(ByVal Connector As ConnectorLine,
ByVal Source As Boolean)
    If Source Then
        AnyShape.FillColor = vbGreen
        Connector.LineColor = vbYellow
   Else
        AnyShape.FillColor = vbRed
        Connector.LineColor = vbRed
   End If
   ' Point igxShape at the same object as AnyShape
   Set igxShape = AnyShape.PermanentShape
End Sub
```

See Also

PermanentConnectorLine property

PermanentDepartment property

PermanentDiagram property

PermanentDiagramObject property

PermanentDocument property

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

Replace Method

Syntax *Shape*.**Replace**(*ShapeTemplate* As ShapeLibraryItem)

Description The Replace method replaces the current shape with the specified shape library item. The *ShapeTemplate* argument specifies the shape library item to use.

Example The following example replaces an existing shape in a diagram with a shape from a Shape Library.

```
' Dimension the variables
Dim igxShape As Shape
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "Shape created. Click OK to Replace it"
' Replace the shape with a ShapeLibrary item
igxShape.Replace ShapeLibraries.Item(1).Item(3)
MsgBox "View the result"
```

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

RevertToShapeClass Method

Syntax *Shape*.**RevertToShapeClass**

Description The RevertToShapeClass method reverts a shape back to the original settings specified by the shape class. In particular, this method reverts the shape back to the original graphic and the original text layout.

Example The following example creates a shape, and then replaces it's graphic with a new graphic. It then invokes the RevertToShape method, which restores the original graphic in the shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim igxGraphicBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a rectangle to the graphic
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Set the color of the rectangle to red
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
' Add an ellipse to the graphic
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the color of the ellipse to blue
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Add a 5 point star to the graphic
igxGraphicBuilder.Star 0.5, 0.5, 0.3, 0.15, 5, 30
' Set the color of the star to green
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(3). _
    FillFormat.FillColor = vbGreen
' Replace the graphic inside the shape with the new graphic
MsgBox "Click OK to replace the graphic."
igxShape.Graphic.Replace igxGraphicBuilder.Graphic
MsgBox "Click OK to invoke RevertToShapeClass."
'Revert the shape back to the shape class that created it
igxShape.RevertToShapeClass
MsgBox "Click OK to continue."
```

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

SetLink Event

Syntax **Private Sub** *Shape*_**SetLink**(*Link* As Link)

Description The SetLink event occurs when a link is added to the Links collection of the specified Shape object, or when an existing link is modified. The *Link* parameter returns a Link object that represents the link that has been added or modified.

Example The following example sets up the SetLink event to listen for links being added. If a link is added to any shape, it displays a message that a new link has been created.

```
Public Sub Test()
    ' Dimension the variables
   Dim igxDiagram1 As Diagram
   Dim igxShape As Shape
   Dim igxLinks As Links
   ' Set the igxDiagram variable to the active diagram object
   Set igxDiagram1 = Application.ActiveDiagram
   ' Create the shape on the active diagram
   Set igxShape = igxDiagram1.DiagramObjects.AddShape(1440, 1440)
   ' Get the Links collection from the shape
   Set igxLinks = igxShape.Links
   ' Add a new link to the shape
   MsgBox "Shape created. Click OK to add a link to the shape."
   igxLinks.AddDiagramLink "Diagram2"
   igxLinks.Item(1).Description = "Diagram2"
End Sub

Private Sub AnyShape_SetLink(ByVal Link As Link)
   MsgBox "A new link was just created"
End Sub
```

See Also Link object

Links object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ShadowColor Property

Syntax *Shape*.**ShadowColor**

Data Type Color (read/write)

Description The ShadowColor property specifies the color to use for the shadow of a shape. The property is valid only if the ShadowType property is not set to ixShadowNone.

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The shadow properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a shadow property most recently is the one that is used.

For more information about shadow properties, refer to the iGrafx Professional User's Guide, or the Format—Shadow/3D dialog.

Example The following example creates a shape on the active diagram, then adds a gray shadow with a depth of 5 to the shadow.

```
' Dimension the variables
Dim igxShape As Shape
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a shadow to the shape
MsgBox "Click OK to add a shadow"
igxShape.ShadowType = ixShadow17
' Set the color of the shadow to gray
MsgBox "Click OK to color the shadow gray."
igxShape.ShadowColor = RGB(100, 100, 100)
' Set the depth of the shadow to 5
MsgBox "click OK to increase shadow depth to 5."
igxShape.ShadowDepth = 5
MsgBox "Click OK to continue"
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ShadowDepth Property

Syntax *Shape*.**ShadowDepth**

Data Type Long (read/write)

Description The ShadowDepth property specifies the depth (or length) of the shadow for a shape; that is, how far the shadow appears to extend away from the shape. Valid values for this property can be between 1 and 5, inclusive. These values represent fixed lengths based on the type of shadow applied to the shape.

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The shadow properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a shadow property most recently is the one that is used.

For more information about shadow properties, refer to the iGrafx Professional User's Guide, or the Format—Shadow/3D dialog.

Example The following example creates a shape on the active diagram, then adds a gray shadow with a depth of 5 to the shadow.

```
' Dimension the variables
Dim igxShape As Shape
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a shadow to the shape
MsgBox "Click OK to add a shadow"
igxShape.ShadowType = ixShadow17
' Set the color of the shadow to gray
MsgBox "Click OK to color the shadow gray."
igxShape.ShadowColor = RGB(100, 100, 100)
' Set the depth of the shadow to 5
MsgBox "click OK to increase shadow depth to 5."
igxShape.ShadowDepth = 5
MsgBox "Click OK to continue"
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ShadowType Property

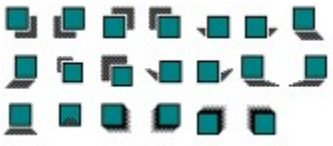
Syntax *Shape*.**ShadowType**

Data Type IxShadowType enumerated constant (read/write)

Description The ShadowType property specifies the type of shadow effect to apply to a shape.

If this property is set to a value other than ixShadowNone, it overrides the ThreeDType property. Conversely, if the ThreeDType property is set to a value other than ixThreeDNone, it overrides the ShadowType property.

The various shadow effects are shown below:



The IxShadowType constant defines valid values for this property, which are listed in the following table.

Value	Name of Constant	Description	
0	ixShadowNone	No shadow.	(Default)
1	ixShadow1		
2	ixShadow2		
3	ixShadow3		
4	ixShadow4		
5	ixShadow5		
6	ixShadow6		
7	ixShadow7		
8	ixShadow8		
9	ixShadow9		
10	ixShadow10		
11	ixShadow11		
12	ixShadow12		
13	ixShadow13		
14	ixShadow14		
15	ixShadow15		
16	ixShadow16		
17	ixShadow17		
18	ixShadow18		
19	ixShadow19		
20	ixShadow20		

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The shadow properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a shadow property most recently is the one that is used.

For more information about shadow properties, refer to the iGrafx Professional User's Guide, or

the Format—Shadow/3D dialog.

Example The following example creates a shape on the active diagram, then adds a gray shadow with a depth of 5 to the shadow.

```
' Dimension the variables
Dim igxShape As Shape
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a shadow to the shape
MsgBox "Click OK to add a shadow"
igxShape.ShadowType = ixShadow17
' Set the color of the shadow to gray
MsgBox "Click OK to color the shadow gray."
igxShape.ShadowColor = RGB(100, 100, 100)
' Set the depth of the shadow to 5
MsgBox "click OK to increase shadow depth to 5."
igxShape.ShadowDepth = 5
MsgBox "Click OK to continue"
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ShapeClass Property

Syntax *Shape*.**ShapeClass**

Data Type ShapeClass object (read-only, See Object Properties)

Description The ShapeClass property returns the ShapeClass object for the specified Shape object. The ShapeClass object allows you to manipulate such properties as connection points, text block, height, and width of a shape. This is useful when you want to adjust the attributes of a shape in a shape library.

The ShapeClass property returns the ShapeClass object for the specified Shape object. For more information, see the ShapeClass object.

Example The following example creates a shape on the active diagram, and gets the ShapeClass object. It then modifies the shape class by reducing the width. Then two more shapes are created, and their appearance reflects the modified shape class.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShapeClass As ShapeClass
' Create the shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeClass object of the first shape
Set igxShapeClass = igxShape1.ShapeClass
MsgBox "Shape created. Click OK to shrink the width of " _
    & "the shape class."
' Reduce the width of the shape class
igxShapeClass.Width = 720
Application.RefreshUI
MsgBox "ShapeClass changed. Now click OK to create two more shapes."
' Add two more shapes. They will reflect the modified shape class
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 5)
MsgBox "Click OK to revert to original shape class settings"
' Revert the shapes to the original shape class settings
igxShape1.RevertToShapeClass
igxShape2.RevertToShapeClass
igxShape3.RevertToShapeClass
MsgBox "Click OK to continue"
```

See Also ShapeClass object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ShapeFormat Property

Syntax *Shape*.**ShapeFormat**

Data Type ShapeFormat object (read-only, See Object Properties)

Description The ShapeFormat returns the ShapeFormat object for the specified Shape object. The ShapeFormat object allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The same properties can be modified by other properties of the Shape object such as LineColor, LineWidth, etc.

Example The following example creates a shape in the active diagram, and then gets its ShapeFormat object. It then uses the ShapeFormat object to change the fill color of the shape to green.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeFormat As ShapeFormat
' Create the shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeFormat object for the shape
Set igxShapeFormat = igxShape.ShapeFormat
' Change the border line color to green
MsgBox "Click OK to change the ShapeFormat color"
igxShapeFormat.FillFormat.FillColor = vbGreen
MsgBox "Click OK to continue"
```

See Also

BackColor property

FillColor property

FillType property

GradientIndex property

LineColor property

LineStyle property

LineWidth property

PatternIndex property

ShadowColor property

ShadowDepth property

ShadowType property

ThreeDDepth property

ThreeDType property

ShapeFormat object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ShapeNumber Property

Syntax *Shape*.**ShapeNumber**

Data Type ShapeNumber object (read-only, See Object Properties)

Description The ShapeNumber property returns the ShapeNumber object for the specified Shape object. Even though the shape number may not be visible on a shape, every shape has a shape number. You can make the shape number visible by setting the ShowNumbering property to True.

Example The following example creates a new shape in the active diagram, and then gets the ShapeNumber object from the shape. It then uses the ShapeNumber object to output the formatted shape number to the Output window.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeNumber As ShapeNumber
' Create the shape in the active diagram
MsgBox "Click OK to create a new shape."
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeNumber object form the shape
Set igxShapeNumber = igxShape.ShapeNumber
' Display the formatted ShapeNumber in a message
MsgBox "The ShapeNumber of the new shape is " _
    & igxShapeNumber.FormattedValue
```

See Also ShapeNumber object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ShowNumbering Property

Syntax *Shape*.**ShowNumbering**[= {**True** | **False**}]

Data Type Boolean (read/write)

Description The ShowNumbering property specifies whether the shape numbering is visible on the diagram for a shape. Shapes are numbered in creation order initially, but the ordering can be changed (see the ShapeNumber object). Setting this property to True causes shape numbers to be shown on the shape.

Example The following example creates a shape in the active diagram, and then shows the shape's number.

```
' Dimension the variables
Dim igxShape As Shape
' Create the shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Activate the shape numbering for the shape
MsgBox "Shape created. Click OK to show it's number."
igxShape.ShowNumbering = True
MsgBox "Click OK to continue."
```

See Also ShapeNumber object

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

StartPointName Property

Syntax *Shape*.**StartPointName**

Data Type String (read/write)

Description The StartPointName property specifies the start point name for the specified Shape object. If the shape is not a start point, this property returns an error.

Start points specify shapes within the same diagram or in different diagrams in the same document (subprocesses) where execution can start when using iDiagrams. You can then set links from one shape to another, if the shape is a start point. The executing entity jumps to the linked start point shape and continues executing.

Use the IsStartPoint property to test whether a shape is a start point. You make a shape a start point by assigning a name to the StartPointName property. You can remove the start point designation from a shape by setting the IsStartPoint property of the shape to False.

Error If you try to read this property when it does not contain a value, an IGRAFX_E_NOTASTARTPOINT error is generated.

Example The following example creates a shape in the active diagram, makes it a start point, and sets the name of the start point.

```
' Dimension the variables
Dim igxDiagramObjects As DiagramObjects
Dim igxDiagramObject As DiagramObject
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Make the shape a start point by assigning a name to the
' StartPointName property
igxShape.StartPointName = "Sub-Process 1"
' Create a second shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
' Find the shapes that are start points
For Each igxDiagramObject In igxDiagramObjects
    igxDiagramObject.Selected = True
    ' If shape is a starting point, display a message
    If igxDiagramObject.Shape.IsStartPoint Then
        MsgBox "This shape is a start point. Links can jump to it." _
            & Chr(13) & "The start point name is " _
            & igxShape.StartPointName
    Else
        MsgBox "This shape is not a start point"
    End If
    igxDiagramObject.Selected = False
Next igxDiagramObject
```

See Also IsStartPoint property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

TextBlock Property

Syntax *Shape*.**TextBlock**

Data Type TextBlock object (read-only, See Object Properties)

Description The TextBlock property returns the TextBlock object for the specified Shape object. For detailed information about the use of text blocks, refer to the iGrafx Professional User's Guide.

Example The following example creates a shape in the active diagram, and then gets the TextBlock object from the shape. It then uses the TextBlock object to orient the shape's text block at 90 degrees.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlock As TextBlock
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the shape's TextBlock object
Set igxTextBlock = igxShape.TextBlock
' Add some text in the shape
igxShape.Text = "My Activity"
MsgBox "Shape created with text. Click OK to orient the " _
    & "text block 90 degrees."
' Orient the text block 90 degrees
igxShape.TextBlock.BlockFormat.Orientation = ixOrientation90
MsgBox "Click OK to continue"
```

See Also

TextBlock object

ChildTextBlock object

ChildTextBlocks object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

TextRTF Property

Syntax *Shape*.**TextRTF**

Data Type String (write-only)

Description The TextRTF property is an alternative to the Text or TextLF properties for specifying the text of the shape. The property allows you to specify a text string that uses Rich Text Formatting embedded commands. Note that you can you write this property; you cannot read it.

Example The following example creates a shape in the active diagram, and then sets the shape's text using the TextRTF property.

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ThreeDDepth Property

Syntax *Shape*.**ThreeDDepth**

Data Type Integer (read/write)

Description The ThreeDDepth property specifies the depth of the three-dimensional effect for the Shape object. Valid values for this property are from 1 to 5. If a value less than 1 is supplied, then 1 is used. If a value greater than 5 is supplied, then 5 is used.

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The ThreeD properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a ThreeD property most recently is the one that is used.

For more information about three dimensional effects, refer to the iGrafx Professional User's Guide, or the Format—Shadow/3D dialog.

Example The following example creates a shape in the active diagram, and then sets the three dimensional style and depth for the shape.

```
' Dimension the variables
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "Click OK to set the shape's 3D format."
' Set the ThreeDType of the shape to ixThreeD1
igxShape.ThreeDType = ixThreeD1
' Set the ThreeDDepth to 3
igxShape.ThreeDDepth = 3
MsgBox "Click OK to continue."
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ThreeDType Property

Syntax *Shape*.**ThreeDType**

Data Type IxThreeDType enumerated constant (read/write)

Description The ThreeDType property specifies the type of three-dimensional effect to apply to the Shape object.

If this property is set to a value other than ixThreeDNone, it overrides the ShadowType property. Conversely, if the ShadowType property is set to a value other than ixShadowNone, it overrides the ThreeDType property.

The various 3D effects are shown below:

The IxThreeDType constant defines valid values for this property, which are listed in the following table.

.

Value	Name of Constant	Description
0	ixThreeDNone	No three dimensional effect.
1	ixThreeD1	
2	ixThreeD2	
3	ixThreeD3	
4	ixThreeD4	
5	ixThreeD5	
6	ixThreeD6	
7	ixThreeD7	
8	ixThreeD8	
9	ixThreeD9	
10	ixThreeD10	
11	ixThreeD11	
12	ixThreeD12	
13	ixThreeD13	
14	ixThreeD14	
15	ixThreeD15	
16	ixThreeD16	

This functionality is also contained in the ShapeFormat object, which allows you to set a shape's formats for line and fill types, and shadow and 3D effects. The ThreeD properties at the Shape object level have the same precedence as those at the ShapeFormat object level. That is, whichever object sets a ThreeD property most recently is the one that is used.

For more information about three dimensional effects, refer to the iGrafx Professional User's Guide, or the Format—Shadow/3D dialog.

Example The following example creates a shape in the active diagram, and then sets the three

dimensional style and depth for the shape.

```
' Dimension the variables
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "Click OK to set the shape's 3D format."
' Set the ThreeDType of the shape to ixThreeD1
igxShape.ThreeDType = ixThreeD1
' Set the ThreeDDepth to 3
igxShape.ThreeDDepth = 3
MsgBox "Click OK to continue."
```

See Also ShapeFormat property

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

TopDepartment Property

Syntax *Shape*.**TopDepartment**

Data Type Department object (read-only, See Object Properties)

Description The TopDepartment property returns the Department object that is the first (or top-most) department on the diagram to which the specified shape belongs. When a shape is placed in a diagram within the boundaries of a department, the TopDepartment and BottomDepartment properties are filled in automatically. The property returns an object that contains "Nothing" if there is no top department for the shape.

Setting the TopDepartment and the BottomDepartment properties causes iGrafx Professional to stretch the shape from the top of the TopDepartment to the bottom of the BottomDepartment. All departments that a shape is drawn in are listed in the DepartmentRange collection for the shape. This information, along with the shape's ExcludedDepartmentNames property, position a shape relative to the departments in a diagram.

Example The following example creates three departments, and then places a shape so it is located within the boundaries of the first department. It then tests to determine whether the TopDepartment and BottomDepartment properties were set automatically when the shape was created. It then changes the value of each property to show that the shape expands and moves based on the value of these properties.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDepartment1 As Department
Dim igxDepartment2 As Department
Dim igxDepartment3 As Department
' Create three departments
Set igxDepartment1 = _
    ActiveDiagram.Departments.AddDepartment("TestDept1")
Set igxDepartment2 = _
    ActiveDiagram.Departments.AddDepartment("TestDept2")
Set igxDepartment3 = _
    ActiveDiagram.Departments.AddDepartment("TestDept3")
' Create a new shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 1.5, 1440)
' Test whether placement of the shape sets the top and bottom
' department properties for the shape
If (igxShape.TopDepartment = "Nothing") Then
    MsgBox "Top department property is not set." & Chr(13) _
        & "Set it to TestDept1."
    igxShape.TopDepartment = igxDepartment1
Else
    MsgBox "Top department property was set on placement. " _
        & "It is: " & igxShape.TopDepartment.DepartmentName
End If
If (igxShape.BottomDepartment = "Nothing") Then
    MsgBox "Bottom department property is not set." & Chr(13) _
        & "Set it to TestDept1."
    igxShape.BottomDepartment = igxDepartment1
Else
    MsgBox "Bottom department property was set on placement. " _
        & "It is: " & igxShape.BottomDepartment.DepartmentName
End If
```

```
' Change the top and bottom departments for the shape
igxShape.BottomDepartment = igxDepartment3
MsgBox "View the result."
igxShape.TopDepartment = igxDepartment2
MsgBox "View the result."
```

See Also

BottomDepartment property

DepartmentRange property

ExcludedDepartmentNames property

Department object

iGrafx API Object Hierarchy

{button Shape object,JI(`igrafxrf.HLP',`Shape_Object')}

ShapeClass Object

The ShapeClass object stores attributes that are common to all the shapes of a particular type. For example, if you have 100 "Decision" shapes in a diagram, they all share the same ShapeClass. The ShapeClass associates a default graphic, connect points, and text layout with all the shapes of a particular type in a diagram.

VBA code also can be associated with a ShapeClass through the VBA Shape Project. To create a VBA Shape Project, you click on a shape, then choose the Code Assistant from the VBA toolbar. Pick an event you want to handle and choose "For all shapes of this type." Doing this creates a VBA Shape Project that is associated with the ShapeClass for the shape you chose (and consequently, all other shapes of that type). VBA Shape Projects and the ShapeClass object form the platform for creating iShapes.

Replacing a graphic at the ShapeClass level changes all shapes in a diagram and all future shapes added unless a shape has been modified from its original graphic (adjustments, for instance).

The following code creates a shape in the active diagram, and then gets its ShapeClass object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeClass As ShapeClass
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeClass object for the shape
Set igxShapeClass = igxShape.ShapeClass
```

Properties, Methods, and Events

All of the properties, methods, and events for the ShapeClass object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		Initialize
ClassID		
ConnectPoints		
Graphic		
Height		
Instances		
Name		
Parent		
TextBlock		
Width		

ClassID Property

Syntax *ShapeClass*.**ClassID**

Data Type String (read-only)

Description The ClassID property returns the Guaranteed Unique Identifier (GUID) for the specified ShapeClass object. The ClassID is used to identify the ShapeClass, and can be used when comparing ShapeClass objects. The ClassID is generated by iGrafx Professional.

Example The following example determines if two shapes in a diagram belong to the same ShapeClass.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShapeClass1 As ShapeClass
Dim igxShapeClass2 As ShapeClass
' Create two shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
Set igxShapeClass1 = igxShape1.ShapeClass
Set igxShapeClass2 = igxShape2.ShapeClass
MsgBox "View the diagram"
If igxShapeClass1.ClassID = igxShapeClass2.ClassID Then
    MsgBox "These two shapes come from the same shape class"
Else
    MsgBox "These two shapes are of different shape classes"
End If
' Create two more shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 2, 1440, Application.ShapeLibraries.Item(1).Item(1))
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 2, 1440 * 3, Application.ShapeLibraries.Item(1).Item(3))
Set igxShapeClass1 = igxShape1.ShapeClass
Set igxShapeClass2 = igxShape2.ShapeClass
MsgBox "View the diagram"
If igxShapeClass1.ClassID = igxShapeClass2.ClassID Then
    MsgBox "These two shapes come from the same shape class"
Else
    MsgBox "These two shapes are of different shape classes"
End If
```

{button ShapeClass object,JI(`igrafxrf.HLP',`ShapeClass_Object')}

ConnectPoints Property

Syntax *ShapeClass*.**ConnectPoints**

Data Type ConnectPoints object (read-only, See Object Properties)

Description The ConnectPoints property returns the ConnectPoints object for the specified ShapeClass object. The connect points are the green points on a shape that become visible as a connector line is moved around a shape. The connect points indicate predefined points at which a connector line can be attached to a shape.

In addition to connect points, iGrafx Professional supports an "all points" connection mode that allows you to connect to other points on the shape besides the green connect points.

Example The following example creates a shape oi the active diagram, and then gets its ShapeClass object. It then uses the ConnectPoints collection to add two new connect points to the shape.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShapeClass As ShapeClass
Dim igxConnectPoints As ConnectPoints
' Create two shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
' Get the ShapeClass object for the shape
Set igxShapeClass = igxShape1.ShapeClass
' Get the ConnectPoints object
Set igxConnectPoints = igxShapeClass.ConnectPoints
' Add two new connect points to the shape class
MsgBox "Click OK to add connect points to the ShapeClass"
igxConnectPoints.Add 0.25, 0
igxConnectPoints.Add 0.75, 0
MsgBox "Points added. Return to the diagram and try" _
    & Chr(13) & "dragging a connector line to the shapes."
```

See Also ConnectPoint object

ConnectPoints object

iGrafx API Object Hierarchy

{button ShapeClass object,JI(`igrafxrf.HLP',`ShapeClass_Object')}

Graphic Property

Syntax *ShapeClass*.**Graphic**

Data Type Graphic object (read-only, See Object Properties)

Description The Graphic property returns the Graphic object for the specified ShapeClass object. The graphic object can be used to modify the graphical representation of the shape class (the 'Process' shape, for instance), and which is shared by all instanced shapes.

For example,

Example The following example creates two shapes of the same shape class, and then gets the ShapeClass object of the first shape. Using the Graphic property, the graphic of the ShapeClass is changed, which changes the graphic in all the shapes of that class.

```
' Dimension the variables
Dim igxShapeClass As ShapeClass
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxGraphic As Graphic
Dim igxGraphicBuilder As New GraphicBuilder
' Create two shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
' Get the ShapeClass from the first shape
Set igxShapeClass = igxShape1.ShapeClass
' Add a rectangle to the GraphicBuilder
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Set the color of the rectangle to red
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
' Add an ellipse to the GraphicBuilder
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
' Add a 5 point star to the GraphicBuilder
igxGraphicBuilder.Star 0.5, 0.5, 0.3, 0.15, 5, 30
' Replace the graphic inside the shape with the new graphic
MsgBox "Click OK to replace the graphic in the ShapeClass."
igxShapeClass.Graphic.Replace igxGraphicBuilder.Graphic
MsgBox "Click OK to continue."
```

See Also Graphic object

iGrafx API Object Hierarchy

{button ShapeClass object,JI(`igrafxrf.HLP',`ShapeClass_Object')}

Initialize Event

Syntax **Private Sub** *ShapeClass*_**Initialize**()

Description The Initialize event occurs when the specified ShapeClass object is first created in a diagram. This event typically occurs when a new type of shape is added to a diagram (resulting in a new shape class being added to the diagram).

This is a useful if there are actions that need to occur when a shape class is initialized, such as setting up custom property lists at the Diagram level, etc.

Example The following example creates a new shape in the active diagram. This triggers the Initialize event.

```
Public WithEvents igxShapeClass As ShapeClass

Public Sub Main()
   Set igxShapeClass = ActiveDiagram.DiagramType. _
       ShapeLibrary.Item(1).ShapeClass
   ActiveDiagram.DiagramObjects.AddShape 1440, 1440, _
       ActiveDiagram.DiagramType.ShapeLibrary.Item(1)
End Sub

Public Sub igxShapeClass_Initialize()
MsgBox "A new ShapeClass, " _
   & ActiveDiagram.DiagramType.ShapeLibrary.Item(1).ShapeClass.Name _
   & " has been initialized."
End Sub
```

{button ShapeClass object,JI(`igrafxrf.HLP',`ShapeClass_Object')}

Instances Property

Syntax *ShapeClass*.**Instances**(*Diagram* As Diagram) As ObjectRange

Data Type ObjectRange object (read-only, See Object Properties)

Description The Instances property returns an ObjectRange object for a ShapeClass object in the diagram specified by the *Diagram* argument. This object range contains all of the objects that contain instances of the shape class. The *Diagram* argument specifies in which diagram to look for the instances of the shape class.

Example The following example adds three shapes to the diagram (of the same type, so they share a shape class). It then gets the ShapeClass object from one of the shapes, and displays the names of all the shapes in the diagram that are members of that ShapeClass by using the Instances property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShapeClass As ShapeClass
Dim igxObjectRange As ObjectRange
' Create a shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 5)
' Give names to the shapes
igxShape1.DiagramObject.ObjectName = "Activity A"
igxShape2.DiagramObject.ObjectName = "Activity B"
igxShape3.DiagramObject.ObjectName = "Activity C"
' Get the ShapeClass from the first shape
Set igxShapeClass = igxShape1.ShapeClass
Set igxObjectRange = igxShapeClass.Instances(ActiveDiagram)
' Collect the object names of the objects in the ObjectRange
For Index = 1 To igxObjectRange.Count
   sNames = sNames & igxObjectRange.Item(Index).ObjectName & Chr(13)
Next Index
' Display the results
MsgBox "These shapes contain Instances of the ShapeClass:" _
   & Chr(13) & Chr(13) & sNames
```

See Also ObjectRange object

iGrafx API Object Hierarchy

{button ShapeClass object,JI(`igrafxrf.HLP',`ShapeClass_Object')}

TextBlock Property

Syntax *ShapeClass*.**TextBlock**

Data Type TextBlock object (read-only, See Object Properties)

Description The TextBlock property returns the TextBlock object for the specified ShapeClass object. The TextBlock property specifies the initial text layout for a shape of the specified type. However, the shape can override the ShapeClass settings and use its own settings, should the user choose to edit the shape's text layout.

Example The following example places a shape in the diagram from the diagram's shape library. Next, the shape's ShapeClass is accessed to get to the TextBlock object for the shape class. The main text block margins are moved in from the shape border, and the boundary is changed to a solid red line. Next, a child text block is added at the top of the text block, and its border is changed to a dashed blue line. Finally, a new ShapeLibraryItem of the same class as the first is added, showing that the changes made to the text blocks of the shape class are used for all new shapes of that class.

```
' Dimension the variables
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraryItem As ShapeLibraryItem
Dim igxShapeClass As ShapeClass
Dim igxShape As Shape
Dim igxTextBlock As TextBlock
Dim igxChildTextBlock  As ChildTextBlock
' Get the ShapeLibrary from the DiagramType object
' of the Diagram object
Set igxShapeLibrary = ActiveDiagram.DiagramType.ShapeLibrary
' Get the first item from the ShapeLibrary
Set igxShapeLibraryItem = igxShapeLibrary.Item(1)
' Add the shape library item to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, igxShapeLibraryItem)
MsgBox "View the diagram"
' Get the shape class from the ShapeLibraryItem
Set igxShapeClass = igxShapeLibraryItem.ShapeClass
' Get the TextBlock object from the shape class
Set igxTextBlock = igxShapeClass.TextBlock
' Set text block properties
With igxTextBlock
    .LeftMargin = 0.1
    .RightMargin = 0.1
    .BottomMargin = 0.1
    .TopMargin = 0.1
    .BlockFormat.LineFormat.Color = vbRed
    .Text = "Main"
End With
MsgBox "View the diagram"
' Add a new child text block to the top of the shape
Set igxChildTextBlock = igxTextBlock.ChildTextBlocks.Add _
    (ixTextTop, 300)
' Set child text block properties
With igxChildTextBlock
    .BlockFormat.LineFormat.Style = ixLineDashDot
    .BlockFormat.LineFormat.Color = vbBlue
```

```
    .Text = "Child"
End With
MsgBox "View the diagram"
' Add another shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3, igxShapeLibraryItem)
MsgBox "View the diagram"
```

See Also

TextBlock object

iGrafx API Object Hierarchy

{button ShapeClass object,JI(`igrafxrf.HLP',`ShapeClass_Object')}

ShapeLibrary Object

The ShapeLibrary object is a collection of ShapeLibraryItem objects (shapes). A ShapeLibrary represents a Share Media palette (displayed in the gallery), or a DiagramType object's document-level ShapeLibrary (accessible through the user interface using File, ShapeLibrary). Using the ShapeLibrary object, you can add, delete, retrieve, and edit items in a palette.

If you get a ShapeLibrary object from the Application's ShapeLibraries object, you are accessing shapes that are stored within Share Media. If you get the ShapeLibrary object from a DiagramType in a document, you are accessing shapes in the document's ShapeLibrary, which is stored in the document. There is one document-stored ShapeLibrary for each DiagramType in a document.

The following example adds a ShapeLibraryItem from a Share Media palette to a document ShapeLibrary.

```
ActiveDiagram.DiagramType.ShapeLibrary.Add(ShapeLibraries.Item(1).Item(1))
```

The properties and methods of the ShapeLibrary object allow you to:

- Add a Shape or a Graphic to the library as a ShapeLibraryItem object (a shape).
- Access any ShapeLibraryItem in the ShapeLibrary, and find out how many items arte in the ShapeLibrary.
- Retrieve the collection name and subject name of a ShapeLibrary.
- Select a shape contained in the ShapeLibrary.
- Close a ShapeLibrary.

NEW

ShapeLibraryItems in a DiagramType's shape library are reference counted. What this means is that if no shapes are using that shape library item, the shape library item will automatically be deleted. This is to reduce clutter and save space in the document stored shape libraries. To keep a shape library item in a document, even when no shapes are using it, you need to make sure "ShowInToolbar" is set to true. Otherwise, the shape library item will be automatically removed.

Properties, Methods, and Events

All of the properties, methods, and events for the ShapeLibrary object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
CollectionName	AddFromGraphic	
Count	AddFromShape	
Parent	Close	
SubjectName	Item	
	SelectShape	

Add Method

Syntax *ShapeLibrary*.**Add**(*Shape* As ShapeLibraryItem, [*AddUnique* As Boolean = True])

Description The Add method adds a ShapeLibraryItem (a shape) to the specified ShapeLibrary object. This is useful when you are creating specialized shape libraries from existing shape library items. The method's arguments are described below.

The *Shape* argument is a ShapeLibraryItem that represents the item that is to be added to the shape library.

The *AddUnique* argument is a Boolean value that specifies whether to add a unique version of the shape specified by the *Shape* argument to the shape library. If you have a collection of shapes that are all the same type, and you set *AddUnique* to False, the shape library knows they are all the same and only adds one of them. If you want all of the items added, set this value to True (or exclude the argument) so that unique versions of the item are added to the shape library. This argument is optional, and defaults to True.

Example The following example gets the ShapeLibrary object from the active diagram and then adds a new shape to the ShapeLibrary from an application level ShapeLibrary (iGrafx Share Media). It also sets the ShowInToolbar property to True for the ShapeLibraryItem object so that it is visible in the toolbar.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxDiagramShapeLibrary As ShapeLibrary
Dim igxAppShapeLibrary As ShapeLibrary
Dim igxAppShapeLibraryItem As ShapeLibraryItem
' Get the DiagramType object
Set igxDiagramType = ActiveDiagram.DiagramType
' Get the ShapeLibrary object at the diagram level
Set igxDiagramShapeLibrary = igxDiagramType.ShapeLibrary
' Get the first ShapeLibrary from the Application object
Set igxAppShapeLibrary = Application.ShapeLibraries.Item(1)
' Get the second shape from the ShapeLibrary
Set igxAppShapeLibraryItem = igxAppShapeLibrary.Item(2)
' Make the ShapeLibraryItem visible in the toolbar
igxAppShapeLibraryItem.ShowInToolbar = True
' Add the shape to the diagram type ShapeLibrary
igxDiagramShapeLibrary.Add igxAppShapeLibraryItem, False
MsgBox "Click OK to continue."
```

See Also AddFromGraphic method

AddFromShape method

{button ShapeLibrary object,JI(`igrafxrf.HLP',`ShapeLibrary_Object')}

AddFromGraphic Method

Syntax

ShapeLibrary.**AddFromGraphic**(*Graphic* As Graphic, [*Name* As String], [*Width* As Integer], [*Height* As Integer], [*AddUnique* As Boolean = True]) As ShapeLibraryItem

Description

The AddFromGraphic method creates a new ShapeLibraryItem and adds it to the specified ShapeLibrary. The new ShapeLibraryItem is based on the graphic object specified with the *Graphic* argument. This is useful if you are building customized graphics using the GraphicBuilder object.

Use this method to add a Graphic object to the shape library (see the AddFromShape method to add shape objects to the shape library). Once added to a shape library, the Graphic object becomes a shape.

The arguments for this method are discussed below.

The *Graphic* argument is a Graphic object that represents the graphic to be added to the ShapeLibrary as a new ShapeLibraryItem (a shape). This Graphic object can come from the GraphicBuilder, a Shape, a TextGraphic, an individual Graphic within a GraphicGroup, a ShapeClass's graphic, etc.

The *Name* argument is the name to give to the new ShapeLibraryItem. This argument is optional.

The *Width* argument is an integer value that represents the default width of the ShapeLibraryItem. When the user creates a shape of this type, the shape gets this width by default. The unit of measure is twips (1440 = 1 inch). This argument is optional.

The *Height* argument is an integer value that represents the default height of the ShapeLibraryItem. When the user creates a shape of this type, the shape gets this height by default. The unit of measure is twips (1440 = 1 inch). This argument is optional.

The *AddUnique* argument is a Boolean value that, when True, always creates a new shape library item (a shape). If set to False, a new shape library item is created only if there is no existing shape library item with the same graphic. If an existing shape library item ahs the same graphic, then that shape library item is returned as the method's result. This argument is optional, but defaults to True.

Example

The following example creates a graphic using the GraphicBuilder object. It then adds the graphic to the ShapeLibrary of the diagram.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxShapeLibrary As ShapeLibrary
Dim igxGraphicBuilder As New GraphicBuilder
' Get the DiagramType object
Set igxDiagramType = ActiveDiagram.DiagramType
' Get the ShapeLibrary object
Set igxShapeLibrary = igxDiagramType.ShapeLibrary
' Build the graphic to be added to the ShapeLibrary
' Add a rectangle to the graphic
igxGraphicBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Set the color of the rectangle to red
igxGraphicBuilder.Graphic.FillFormat.FillColor = vbRed
' Add an ellipse to the graphic
igxGraphicBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the color of the ellipse to blue
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Add a 5 point star to the graphic
igxGraphicBuilder.Star 0.5, 0.5, 0.3, 0.15, 5, 30
```

```
' Set the color of the star to green
igxGraphicBuilder.Graphic.GraphicGroup.Graphics.Item(3). _
   FillFormat.FillColor = vbGreen
FillFormat.ProtectFillFormat = True
' Add the graphic to the ShapeLibrary
MsgBox "Click OK to add a new graphic to the ShapeLibrary."
Call igxShapeLibrary.AddFromGraphic(igxGraphicBuilder.Graphic, _
   "Green Star", 1440, 1440)
MsgBox "Click OK to continue."
For iCount = 1 To igxShapeLibrary.Count
    MsgBox "Item " & iCount & " in ShapeLibrary has the " _
        & "following Description:" & Chr(13) _
        & igxShapeLibrary.Item(iCount).Description
Next iCount
```

See Also

Add method

AddFromShape method

GraphicBuilder object

{button ShapeLibrary object,JI(`igrafxrf.HLP',`ShapeLibrary_Object')}

AddFromShape Method

Syntax *ShapeLibrary*.**AddFromShape**(*Shape* As Shape, [*Name* As String]) As ShapeLibraryItem

Description The AddFromShape method adds a new ShapeLibraryItem to the specified ShapeLibrary object, where the added shape is based on an existing shape in a diagram. The new ShapeLibraryItem copies all of the attributes of the specified shape including its graphic, properties, size, and formatting.

The *Shape* argument is a Shape object on which the new ShapeLibraryItem is to be based.

The *Name* argument is the name to give to the new ShapeLibraryItem. This argument is optional.

Example The following example adds a shape to a ShapeLibrary.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxShapeLibrary As ShapeLibrary
Dim igxShape As Shape
' Create a new shape in the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries(1)(3))
' Get the DiagramType object
Set igxDiagramType = ActiveDiagram.DiagramType
' Get the ShapeLibrary object
Set igxShapeLibrary = igxDiagramType.ShapeLibrary
' Add a shape to the ShapeLibrary
MsgBox "Click OK to add a new shape to the ShapeLibrary."
Call igxShapeLibrary.AddFromShape(igxShape)
MsgBox "Click OK to continue."
```

See Also Add method

AddFromGraphic method

{button ShapeLibrary object,JI(`igrafxrf.HLP',`ShapeLibrary_Object')}

Close Method

Syntax *ShapeLibrary*.**Close**()

Description The Close method closes a ShapeLibrary object. The method does not work on a ShapeLibrary that is owned by DiagramType, and an error is displayed if this is attempted.

Error IGRAFX_E_NOTAPPLICABLE is returned if the Close method is called from a ShapeLibrary that is owned by a DiagramType.

Example The following example gets the first ShapeLibrary object from the ShapeLibraries collection of the Application object. It then closes the ShapeLibrary.

```
' Dimension the variables
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraries As ShapeLibraries
' Get the ShapeLibraries object
Set igxShapeLibraries = Application.ShapeLibraries
' Get the first ShapeLibrary from the ShapeLibraries collection
Set igxShapeLibrary = igxShapeLibrary.Item(1)
' Close the ShapeLibrary
MsgBox "Click OK to close the ShapeLibrary."
igxShapeLibrary.Close
MsgBox "Click OK to continue"
```

{button ShapeLibrary object,JI(`igrafxrf.HLP',`ShapeLibrary_Object')}

CollectionName Property

Syntax *ShapeLibrary*.**CollectionName**

Data Type String (read-only)

Description The CollectionName property returns the collection name of the specified ShapeLibrary object. The collection name in iGrafx Share Media is the name of a folder that contains shape palettes). The palette is named by the SubjectName property. A ShapeLibrary should always have both a collection name and a subject name.

There is no way to change the collection name once the ShapeLibrary has been created with the ShapeLibraries.Add method. This property does not work for the ShapeLibrary that is owned by a DiagramType object.

Example The following example gets the ShapeLibraries collection form the Application object. It then goes through the collection and outputs the collection name and subject name of each of the shape libraries in the collection.

```
' Dimension the variables
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraries As ShapeLibraries
' Get the ShapeLibraries object
Set igxShapeLibraries = Application.ShapeLibraries
' Go through the collection and output the collection
' name and subject name to the Output window
For Each igxShapeLibrary In igxShapeLibraries
    Output igxShapeLibrary.CollectionName & " - " _
        & igxShapeLibrary.SubjectName
Next igxShapeLibrary
MsgBox "Click OK to continue."
```

See Also SubjectName property

{button ShapeLibrary object,JI(`igrafxrf.HLP',`ShapeLibrary_Object')}

Item Method

Syntax *ShapeLibrary*.**Item**(*Index* As Integer) As ShapeLibraryItem

Description The Item method returns the ShapeLibraryItem at the specified index in the ShapeLibrary collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type ShapeLibraryItem. The *Index* argument must be a value between 1 and ShapeLibrary.Count, inclusive.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. Use error trapping to handle these errors.

Example The following example iterates through each ShapeLibraryItem in the first application level shape library collection, Application.ShapeLibraries.Item(1). It then prints the description of each ShapeLibraryItem in a message box.

```
' Dimension the variables
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraries As ShapeLibraries
Dim sDescriptions As String
' Get the ShapeLibraries object
Set igxShapeLibraries = Application.ShapeLibraries
' Get the first ShapeLibrary
Set igxShapeLibrary = igxShapeLibraries.Item(1)
' Collect the contents of the ShapeLibrary into string
For Index = 1 To igxShapeLibrary.Count
    sDescriptions = sDescriptions & _
    igxShapeLibrary.Item(Index).Description & Chr(13)
Next Index
' Display the results
MsgBox "The " & igxShapeLibrary.CollectionName & " - " _
    & igxShapeLibrary.SubjectName & _
    " collection contains these shapes:" & Chr(13) & Chr(13) _
    & sDescriptions
```

{button ShapeLibrary object,JI(`igrafxrf.HLP',`ShapeLibrary_Object')}

SelectShape Method

Syntax *ShapeLibrary*.**SelectShape**(*Index*) As Boolean

Description The SelectShape method selects a shape in the specified ShapeLibrary based on the supplied index value (a value between 1 and ShapeLibrary.Count inclusive). This is the same as clicking on a shape in the gallery or toolbar. The result of the method must be assigned to a Boolean variable, which indicates the success or failure of the method.

This method is useful for setting a specific shape to be the one used the next time a shape is added to the diagram. This works both through the user interface, and through VBA using the DiagramObjects.AddShape method.

Example The following example gets the ShapeLibrary object for the Diagram and then selects the first, second, and third shapes from the ShapeLibrary, in order, and adds them to the diagram using the DiagramObjects.AddShape method. A message box indicating success or failure is displayed after each shape is selected. To assure that this example works, open a diagram and add five or six shapes to it from the Share Media palette. Then go to the Shape toolbar for the diagram (typically on the left side of the window), and open the shape library. Check all the shapes to they are added to the toolbar, and then run this code.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDiagramType As DiagramType
Dim igxShapeLibrary As ShapeLibrary
Dim fItemSelected As Boolean
' Get the DiagramType object
Set igxDiagramType = ActiveDiagram.DiagramType
' Get the ShapeLibrary object
Set igxShapeLibrary = igxDiagramType.ShapeLibrary
' Select the first item in the ShapeLibrary
fItemSelected = igxShapeLibrary.SelectShape(1)
' Display a message box indicating success or failure of selection
If fItemSelected Then
    MsgBox ("The first item was selected. Click OK to add it.")
Else
    MsgBox ("The first item was not selected.")
End If
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440)
igxShape.Text = "1"
' Select the second item in the ShapeLibrary
fItemSelected = igxShapeLibrary.SelectShape(2)
' Display a message box indicating success or failure of selection
If fItemSelected Then
    MsgBox ("The second item was selected. Click OK to add it.")
Else
    MsgBox ("The second item was not selected.")
End If
' Now add a new shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3)
igxShape.Text = "2"
' Add a third shape, which is the third item in the diagram's
' Shape Library
```

```
' Select the third item in the ShapeLibrary
fItemSelected = igxShapeLibrary.SelectShape(3)
' Display a message box indicating success or failure of selection
If fItemSelected Then
    MsgBox ("The third item was selected. Click OK to add it.")
Else
    MsgBox ("The third item was not selected.")
End If
' Now add a new shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 3)
igxShape.Text = "3"
MsgBox "End of example. Three different shapes added."
```

See Also ShapeLibraryItem.Select method

{button ShapeLibrary object,JI(`igrafxrf.HLP',`ShapeLibrary_Object')}

SubjectName Property

Syntax *ShapeLibrary*.**SubjectName**

Data Type String (read/write)

Description The SubjectName property returns the subject name of the specified ShapeLibrary object. The subject name in iGrafx Share Media is the name of a shape palette. Shape palettes are grouped under collection names (folders in the Share Media gallery). A ShapeLibrary should always have both a collection name and a subject name.

Unlike the collection name, you can change the subject name once the ShapeLibrary has been created with the ShapeLibraries.Add method. This property does not work for the ShapeLibrary that is owned by a DiagramType object.

Example The following example gets the ShapeLibraries collection from the application object. It then goes through the collection and outputs the collection name and subject name of each of the shape libraries in the collection.

```
' Dimension the variables
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraries As ShapeLibraries
Dim sString As String
' Get the ShapeLibraries object
Set igxShapeLibraries = Application.ShapeLibraries
' Go through the collection and output the collection
' name and subject name in a message box
For Each igxShapeLibrary In igxShapeLibraries
    sString = sString & igxShapeLibrary.CollectionName & " - " _
    & igxShapeLibrary.SubjectName & Chr(13)
Next igxShapeLibrary
MsgBox "All the collections:" & Chr(13) & Chr(13) & sString
```

See Also CollectionName property

{button ShapeLibrary object,JI(`igrafxrf.HLP',`ShapeLibrary_Object')}

ShapeLibraries Object

The ShapeLibraries object is a collection of individual ShapeLibrary objects. A ShapeLibraries collection is only associated with and accessible from the Application object. Its purpose is to store and provide access to the currently open iGrafx Share Media shape palettes.

The ShapeLibraries object provides the following functionality:

- The ability to access any ShapeLibrary object in the collection.
- The ability to determine how many ShapeLibrary objects are in the collection.
- The ability to add a new ShapeLibrary object to the collection.

The ShapeLibraries object provides information about palettes that you currently have open. It does not allow you to browse through iGrafx Share Media palettes that are not open. It does allow you to open a palette that is not currently open by specifying the collection name and subject name.

Properties, Methods, and Events

All of the properties, methods, and events for the ShapeLibraries object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent		

Add Method

Syntax *ShapeLibraries*.**Add**(*Collection* As String, *Subject* As String) As ShapeLibrary

Description The Add method adds an existing ShapeLibrary to the ShapeLibraries collection and returns a ShapeLibrary object that represents the newly added ShapeLibrary. Additionally, you can use it to create a new, empty ShapeLibrary (palette) in iGrafx Share Media.

The *Collection* argument is the collection name to which the ShapeLibrary (palette) being added belongs. If the collection does not already exist, it is created.

The *Subject* argument is the subject name for the ShapeLibrary (palette). If the subject name and collection name match an existing palette, then a new one is not created, but rather the existing one is returned and opened in the Gallery. If the subject name and collection do not exist in iGrafx Share Media, a new palette is created in the specified collection with the specified subject name.

Example The following example gets the ShapeLibraries object from the Application object. It then uses the ShapeLibraries object to add a ShapeLibrary to an already existing collection.

```
' Dimension the variables
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraries As ShapeLibraries
' Get the ShapeLibraries object
Set igxShapeLibraries = Application.ShapeLibraries
' Add a new ShapeLibrary to the ShapeLibraries collection
MsgBox "Click OK to add a new ShapeLibrary to the application."
Set igxShapeLibrary = igxShapeLibraries.Add _
    ("iGrafx Professional Basic Palettes", "New Library")
MsgBox "Click OK to continue."
```

{button ShapeLibraries object,JI(`igrafxrf.HLP',`ShapeLibraries_Object')}

Item Method

Syntax *ShapeLibraries*.**Item**(*Index* As Integer) As ShapeLibrary

Description The Item method returns the ShapeLibrary object at the specified index in the ShapeLibraries collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type ShapeLibrary. The *Index* argument must be a value between 1 and ShapeLibraries.Count, inclusive.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. Use error trapping to handle these errors.

Example The following example displays a message containing all the ShapeLibrary collection and subject names in the application's ShapeLibraries collection.

```
' Dimension the variables
Dim igxShapeLibraries As ShapeLibraries
Dim igxShapeLibrary As ShapeLibrary
Dim sString As String
' Get the ShapeLibraries object
Set igxShapeLibraries = Application.ShapeLibraries
' Go through the collection and output the collection
' name and subject name in a message box
For Index = 1 To igxShapeLibraries.Count
   sString = sString & igxShapeLibraries.Item(Index).CollectionName _
       & " - " & _
       igxShapeLibraries.Item(Index).SubjectName & Chr(13)
Next Index
MsgBox "All the collections:" & Chr(13) & Chr(13) & sString
```

{button ShapeLibraries object,JI(`igrafxrf.HLP',`ShapeLibraries_Object')}

ShapeLibraryItem Object

The ShapeLibraryItem object is an item (a shape) in a ShapeLibrary. A ShapeLibraryItem can be accessed by using the ShapeLibrary collection, which is a collection of ShapeLibraryItem objects.

For example, to access the first ShapeLibraryItem in the ShapeLibrary associated with the active diagram, you would write:

```
Dim MyItem As ShapeLibraryItem
Set MyItem = ActiveDiagram.DiagramType.ShapeLibrary.Item(1)
```

Properties, Methods, and Events

All of the properties, methods, and events for the ShapeLibraryItem object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Commit	
Description	Delete	
Keywords	Select	
Parent		
ShapeClass		
ShowInToolbar		
ToolTip		

Related Topics

ShapeLibrary object

iGrafx API Object Hierarchy

Commit Method

Syntax *ShapeLibraryItem*.**Commit**

Description The Commit method commits any changes made to the specified ShapeLibraryItem object immediately. Changes that you may want to commit immediately include graphic changes, text layout changes, etc. The changes are automatically committed when the variable assigned to the ShapeLibraryItem goes out of scope, but there are still situations where you may need to use Commit explicitly.

Example The following example gets the first ShapeLibraryItem from the ShapeLibrary of the DiagramType object, which is derived from the Diagram object. It then uses the Graphic property of the ShapeClass object to change the fill color of the graphic. Finally it uses the Commit method of the ShapeClass object to apply the changes to the graphic's fill color.

```
' Dimension the variables
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraryItem As ShapeLibraryItem
Dim igxShapeClass As ShapeClass
' Get the ShapeLibrary from the DiagramType object
' of the Diagram object
Set igxShapeLibrary = ActiveDiagram.DiagramType.ShapeLibrary
' Get the first item from the ShapeLibrary
Set igxShapeLibraryItem = igxShapeLibrary.Item(1)
' Get the shape class from the ShapeLibraryItem
Set igxShapeClass = igxShapeLibraryItem.ShapeClass
' Change the fill of the graphic to green
igxShapeClass.Graphic.FillFormat.FillColor = vbGreen
igxShapeClass.Graphic.ProtectFillFormat = True
' Commit the changes to the ShapeLibraryItem
MsgBox "Changes ready. Click OK to Commit."
igxShapeLibraryItem.Commit
MsgBox "Click OK to continue."
```

{button ShapeLibraryItem object,JI(`igrafxrf.HLP',`ShapeLibraryItem_Object')}

Description Property

Syntax *ShapeLibraryItem*.**Description**

Data Type String (read/write)

Description The Description property specifies the description of the specified ShapeLibraryItem object.

Example The following example creates a new Shape and a new ShapeLibrary. It then uses the ShapeLibrary object to set the Description and Keywords properties of the shape.

```
' Dimension the variables
Dim igxShapeLibraries As ShapeLibraries
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraryItem As ShapeLibraryItem
Dim igxShape As Shape
' Create a new ShapeLibrary
Set igxShapeLibrary = _
   Application.ShapeLibraries.Add("MyNewCollection", "Test")
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add the new shape to the new ShapeLibrary
Set igxShapeLibraryItem = igxShapeLibrary.AddFromShape _
   (igxShape, "MyNewShape")
' Set the shape's description and keywords
MsgBox "Click OK to set the shape's description and keywords."
igxShapeLibraryItem.Description = "My new shape"
igxShapeLibraryItem.Keywords = "Process, Test, Box"
' Commit the changes to the ShapeLibraryItem
igxShapeLibraryItem.Commit
MsgBox "Done. Click OK to continue."
```

{button ShapeLibraryItem object,JI(`igrafxrf.HLP',`ShapeLibraryItem_Object')}

Keywords Property

Syntax *ShapeLibraryItem*.**Keywords**

Data Type String (read/write)

Description The Keywords property specifies the keywords that are used to provide additional identifiers for the ShapeLibraryItem object. The keywords in the string should be separated by commas.

Example The following example creates a new Shape and a new ShapeLibrary. It then uses the ShapeLibrary object to set the Description and Keywords properties of the shape.

```
' Dimension the variables
Dim igxShapeLibraries As ShapeLibraries
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraryItem As ShapeLibraryItem
Dim igxShape As Shape
' Create a new ShapeLibrary
Set igxShapeLibrary = _
   Application.ShapeLibraries.Add("MyNewCollection", "Test")
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add the new shape to the new ShapeLibrary
Set igxShapeLibraryItem = igxShapeLibrary.AddFromShape _
   (igxShape, "MyNewShape")
' Set the shape's description and keywords
MsgBox "Click OK to set the shape's description and keywords."
igxShapeLibraryItem.Description = "My new shape"
igxShapeLibraryItem.Keywords = "Process, Test, Box"
' Commit the changes to the ShapeLibraryItem
igxShapeLibraryItem.Commit
MsgBox "Done. Click OK to continue."
```

{button ShapeLibraryItem object,JI(`igrafxrf.HLP',`ShapeLibraryItem_Object')}

Select Method

Syntax *ShapeLibraryItem*.**Select**

Description The Select method selects the specified ShapeLibraryItem (a shape) in the Shape Toolbar of the document, or in the Share Media (a shape pallette), depending on which object you are accessing this method through. Using this method is the same as clicking on a shape in the gallery or toolbar, and basically the same as the ShapeLibrary.SelectShape method.

This method is useful for setting a specific shape to be the one used the next time a shape is added to the diagram. This works both through the user interface, and through VBA using the DiagramObjects.AddShape method.

Example The following example gets the first ShapeLibrary object in the application's ShapeLibraries collection (from the Share Media). It then selects, using the Select method, the first, second, and third shapes from the ShapeLibrary, in order, and adds them to the diagram using the DiagramObjects.AddShape method. A message box is displayed after each shape is selected and added to the diagram.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeLibrary As ShapeLibrary
' Get the the first ShapeLibrary object in the application's
' ShapeLibraries collection--from the Share Media
Set igxShapeLibrary = Application.ShapeLibraries.Item(1)
' Select the first item in the ShapeLibrary
igxShapeLibrary.Item(1).Select
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440)
igxShape.Text = "1"
MsgBox "Added the first shape from the selected " _
    & "Share Media shape library."
' Select the second item in the ShapeLibrary
igxShapeLibrary.Item(2).Select
' Now add a new shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3)
igxShape.Text = "2"
MsgBox "Added the second shape from the selected " _
    & "Share Media shape library."
' Select the third item in the ShapeLibrary
igxShapeLibrary.Item(3).Select
' Now add a new shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 3)
igxShape.Text = "3"
MsgBox "Added the third shape from the selected " _
    & "Share Media shape library."
MsgBox "End of example. Three different shapes added."
```

See Also ShapeLibrary.SelectShape method

{button ShapeLibraryItem object,JI(`igrafxrf.HLP',`ShapeLibraryItem_Object')}

ShapeClass Property

Syntax *ShapeLibraryItem*.**ShapeClass**

Data Type ShapeClass object (read-only, See Object Properties)

Description The ShapeClass property returns the ShapeClass object associated with the specified ShapeLibraryItem object. The ShapeClass object, in turn, provides access to important attributes of the ShapeLibraryItem including the Graphic object, etc.

Example The following example gets the first ShapeLibraryItem from the ShapeLibrary of the DiagramType object, which is derived from the Diagram object. It then uses the Graphic property of the ShapeClass object to change the fill color of the graphic.

```
' Dimension the variables
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraryItem As ShapeLibraryItem
Dim igxShapeClass As ShapeClass
' Get the ShapeLibrary from the DiagramType object
' of the Diagram object
Set igxShapeLibrary = ActiveDiagram.DiagramType.ShapeLibrary
' Get the first item from the ShapeLibrary
Set igxShapeLibraryItem = igxShapeLibrary.Item(1)
' Get the shape class from the ShapeLibraryItem
Set igxShapeClass = igxShapeLibraryItem.ShapeClass
' Change the fill of the graphic to green
igxShapeClass.Graphic.FillFormat.FillColor = vbGreen
igxShapeClass.Graphic.ProtectFillFormat = True
' Commit the changes to the ShapeLibraryItem.
igxShapeLibraryItem.Commit
MsgBox "Click OK to continue."
```

See Also ShapeClass object

iGrafx API Object Hierarchy

{button ShapeLibraryItem object,JI(`igrafxrf.HLP',`ShapeLibraryItem_Object')}

ShowInToolbar Property

Syntax *ShapeLibraryItem*.**ShowInToolbar**[= {**True | False**}]

Data Type Boolean (read/write)

Description The ShowInToolbar property specifies whether the ShapeLibraryItem object is visible in the Standard Toolbar. This method is only valid for a ShapeLibrary object that is owned by a DiagramType. If you set ShowInToolbar to False, the shape is still visible in the Shape toolbar flyout menu, which allows you to add many shapes to the toolbar without the toolbar growing large.

Example The following example gets the ShapeLibrary object from the active diagram and then adds a new shape to the ShapeLibrary from another ShapeLibrary. It also set the ShowInToolbar property to True for the ShapeLibraryItem so that it is visible in the toolbar.

```
'Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxShape As Shape
Dim igxShapeLibrary As ShapeLibrary
Dim igxShapeLibraryItem As ShapeLibraryItem
' Add a new shape to the diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the DiagramType object
Set igxDiagramType = ActiveDiagram.DiagramType
' Get the ShapeLibrary object
Set igxShapeLibrary = igxDiagramType.ShapeLibrary
' Add the shape in the diagram to the ShapeLibrary
Set igxShapeLibraryItem = igxShapeLibrary.AddFromShape(igxShape)
' Make the ShapeLibraryItem visible in the toolbar
MsgBox "Click OK to make the shape visible in the Standard Toobar."
igxShapeLibraryItem.ShowInToolbar = True
' Set the tool tip text for the ShapeLibraryItem
igxShapeLibraryItem.ToolTip = "New Shape"
MsgBox "Click OK to continue"
```

{button ShapeLibraryItem object,JI(`igrafxrf.HLP',`ShapeLibraryItem_Object')}

ToolTip Property

Syntax *ShapeLibraryItem*.**ToolTip**

Data Type String (read/write)

Description The ToolTip property specifies a string of text to display in the tool tip box when the cursor is held over the ShapeLibraryItem.

Example The following example gets the ShapeLibrary object from the active diagram, and then adds a new shape to the ShapeLibrary from another ShapeLibrary. It then sets the ShowInToolbar property to True for the ShapeLibraryItem so that it is visible in the toolbar. Finally, it sets the text for the ToolTip property.

```
' Dimension the variables
Dim igxDiagramType As DiagramType
Dim igxShapeLibrary As ShapeLibrary
Dim igxShape As Shape
Dim igxShapeLibraryItem As ShapeLibraryItem
' Add a new shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the DiagramType object
Set igxDiagramType = ActiveDiagram.DiagramType
' Get the ShapeLibrary object
Set igxShapeLibrary = igxDiagramType.ShapeLibrary
' Add the shape in the diagram to the ShapeLibrary
Set igxShapeLibraryItem = igxShapeLibrary.AddFromShape(igxShape)
' Make the ShapeLibraryItem visible in the toolbar
MsgBox "Click OK to set the Tooltip for the shape."
igxShapeLibraryItem.ShowInToolbar = True
' Set the tool tip text for the ShapeLibraryItem
igxShapeLibraryItem.ToolTip = "New Shape"
MsgBox "Click OK to continue"
```

{button ShapeLibraryItem object,JI(`igrafxrf.HLP',`ShapeLibraryItem_Object')}

ShapeNumber Object

The ShapeNumber object controls and manipulates the number that is associated (or assigned) to a Shape object. It can be used to reposition the number, set its orientation, or manipulate the values it displays. This object is a property only of Shape objects; that is, it can be accessed only through the Shape object.

Shapes can have simple integral numbers, or they can have numbers with multiple parts. Numbers with multiple parts are used in hierarchical numbering schemes.

Note that the ShapeNumber object does not control the formatting of a shape's number. The formatting is handled through the following object tree:

```
Shape.ShapeNumber.Field.FieldText.NumberFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the ShapeNumber object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Trim	
Field		
FormattedValue		
NumberOfSignificantParts		
Parent		
Part		
Shown		
Value		

Related Topics

Shape object

NumberFormat object

Field object

Field Property

Syntax *ShapeNumber*.**Field**

Data Type Field object (read-only, See Object Properties)

Description The Field property returns the Field object that displays the number of a shape. If the shape is not currently displaying a number, the field may not exist. In this case, the property returns the "Nothing" value.

The Field object controls all the formatting options for the shape number, including position, size, orientation, and text formatting.

Example The following example creates a shape on the active diagram and turns on its shape number. It then formats the shape number field so the number is shown centered below the shape, preceded by a number sign.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxNumber As ShapeNumber
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeNumber object form the shape
Set igxNumber = igxShape.ShapeNumber
' Turn on shape numbers for the shape
igxNumber.Shown = True
' Put a number sign before the displayed number
igxNumber.Field.FieldText.NumberFormat.Prefix = "#"
' Set the shape number's position to be below the shape
igxNumber.Field.FieldPosition = ixFieldBelow
MsgBox "Click OK to continue."
```

See Also Field object

FieldText object

NumberFormat object

{button ShapeNumber object,JI(`igrafxrf.HLP',`ShapeNumber_Object')}

FormattedValue Property

Syntax *ShapeNumber*.**FormattedValue**

Data Type String (read-only)

Description The FormattedValue property returns a string containing the shape's number as it is shown on the shape. If the number is not currently showing on the shape, the FormattedValue property returns the shape number string as it would be shown if it were turned on.

The string value is formatted in the style defined by NumberFormat of the FieldText object, which is derived from the Field object. The value of this property depends both on the shape's number and on the formatting information in the shape's number field.

Example The following example creates a shape on the active diagram and turns on its shape number. It then sets up the format for the shape number and moves the shape number's position so that it is centered below the shape. The formatted and unformatted values for the shape number are also output to a message box.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeNumber As ShapeNumber
Dim igxField As Field
Dim igxFieldText As FieldText
Dim igxNumberFormat As NumberFormat
' Create the shape on the active diagram.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeNumber object form the shape
Set igxShapeNumber = igxShape.ShapeNumber
' Turn on shape numbers for the shape
igxShapeNumber.Shown = True
' Get the Field object
Set igxField = igxShapeNumber.Field
' Get the FieldText object
Set igxFieldText = igxField.FieldText
' Get the NumberFormat object
Set igxNumberFormat = igxFieldText.NumberFormat
' Set the format for the shape number. i.e.(X*X*X*X)
igxNumberFormat.NumberOfParts = 4
igxNumberFormat.Separator(1) = "*"
igxNumberFormat.Separator(2) = "*"
igxNumberFormat.Separator(3) = "*"
igxNumberFormat.IncrementingPart = 4
' Set the shape number's position to be below the shape
igxField.FieldPosition = ixFieldBelow
' Output the Formatted value to a message box
MsgBox "The formatted value is " & igxShapeNumber.FormattedValue _
    & Chr(13) & Chr(13) & _
    "The unformatted value is " & igxShapeNumber.Value
```

See Also Value property

{button ShapeNumber object,JI(`igrafxrf.HLP',`ShapeNumber_Object')}

NumberOfSignificantParts Property

Syntax *ShapeNumber*.**NumberOfSignificantParts**

Data Type Integer (read-only)

Description The NumberOfSignificantParts property returns an integer indicating how many parts of the shape number are being used for multi-part shape numbering. Shape numbers are considered to have an unbounded number of parts. This property tells you how many of those parts can be incremented (or are significant), beginning from the left. Number parts that haven't been specified as significant remain at a value of zero.

The following examples illustrate the concept of significant parts for a number.

```
5.0.0.0…          1 Significant Part
6.3.0.0.0…        2 Significant Parts
4.0.0.5.0.0.0…    4 Significant Parts
```

Note that the NumberOfSignificantParts in a shape number has nothing to do with how many parts are actually displayed. The display of a shape number is controlled by the formatting options for the Shape Field that displays the number.

The number of parts is defined by the NumberFormat object which is derived from the FieldText object. The FieldText object is derived from the Field object which can be derived from the ShapeNumber object. The hierarchy is as follows:

```
ShapeNumber.Field.TextField.NumberFormat.NumberOfParts
```

Example The following example creates a new shape with a fomatted number. It then display's the ShapeNumber's number of significant parts.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeNumber As ShapeNumber
Dim igxField As Field
Dim igxFieldText As FieldText
Dim igxNumberFormat As NumberFormat
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeNumber object form the shape
Set igxShapeNumber = igxShape.ShapeNumber
' Turn on shape numbers for the shape
igxShapeNumber.Shown = True
' Get the Field object
Set igxField = igxShapeNumber.Field
' Get the FieldText object
Set igxFieldText = igxField.FieldText
' Get the NumberFormat object
Set igxNumberFormat = igxFieldText.NumberFormat
' Set the format for the shape number. i.e.(X*X*X*X)
igxNumberFormat.NumberOfParts = 4
igxNumberFormat.Separator(1) = "*"
igxNumberFormat.Separator(2) = "*"
igxNumberFormat.Separator(3) = "*"
igxNumberFormat.IncrementingPart = 4
' Set the shape number's position to be below the shape
```

```
igxField.FieldPosition = ixFieldBelow
' Output the Formatted value to a message box
MsgBox "The formatted value is " & igxShapeNumber.FormattedValue _
    & Chr(13) & Chr(13) & _
    "The unformatted value is " & igxShapeNumber.Value
MsgBox "The Formatted number " & igxShapeNumber.FormattedValue _
    & " has " & igxShapeNumber.NumberOfSignificantParts & _
    " significant part(s)."
```

See Also

Field object

FieldText object

NumberFormat object

{button ShapeNumber object,JI(`igrafxrf.HLP',`ShapeNumber_Object')}

Part Property

Syntax *ShapeNumber*.**Part**(*Index* As Integer)

Data Type Long (read/write)

Description The Part property specifies a part of a shape number. The *Index* argument is used to specify which part of the shape number to return or set.

It is important to note that if the format style uses letters instead of numbers, the value returned is still of type Long. For example, if part one shows the letter A, the value returned is 1. This is also true if you set a value. If your format style uses letters, you must set values using numbers only, never letters. If you set a value of 2 and the format is a letter, then you see a B. Values higher than 26 are represented by two or more letters. For example, 27 would be displayed as AA, 28 would be displayed as AB, etc. (Base-26 counting system using only letters.)

Note that when you are using multi-part numbers, the Value property always returns the first part of the number. Additional parts of the number can be accessed only with the Part property.

Important Be aware that the IncrementingPart property of the NumberFormat object does not have any real use through VBA automation, because you can only set that property on a "per shape" basis once a shape is added to a diagram, so it does not help in numbering new shapes that are added. To adjust any additional parts of a shape number through automation, you need to use the Part property to set the value.

Example The following example creates a shape on the active diagram and sets up its shape number. It then modifies the second part of the shape number by changing it to a 2.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeNumber As ShapeNumber
Dim igxField As Field
Dim igxFieldText As FieldText
Dim igxNumberFormat As NumberFormat
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeNumber object form the shape
Set igxShapeNumber = igxShape.ShapeNumber
' Turn on shape numbers for the shape
igxShapeNumber.Shown = True
' Get the Field object
Set igxField = igxShapeNumber.Field
' Get the FieldText object
Set igxFieldText = igxField.FieldText
' Get the NumberFormat object
Set igxNumberFormat = igxFieldText.NumberFormat
' Set the format for the shape number. i.e.(X*X*X*X)
igxNumberFormat.NumberOfParts = 4
igxNumberFormat.Separator(1) = "*"
igxNumberFormat.Separator(2) = "*"
igxNumberFormat.Separator(3) = "*"
igxNumberFormat.IncrementingPart = 4
' Set the shape number's position to be below the shape
igxField.FieldPosition = ixFieldBelow
' Output the old formatted value to a message box
MsgBox "The old formatted value is " & _
    igxShapeNumber.FormattedValue
' Set the second part to two
```

```
igxShapeNumber.Part(2) = 2
' Output the new formatted value to a message box
MsgBox "The new formatted value is " & _
    igxShapeNumber.FormattedValue
```

See Also

Field object

FieldText object

NumberFormat object

{button ShapeNumber object,JI(`igrafxrf.HLP',`ShapeNumber_Object')}

Shown Property

Syntax *ShapeNumber*.**Shown**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Shown property specifies whether the shape number is visible. This is the same as the ShowNumbering property of the Shape object.

Example The following example toggles the Shown state for each shape in the active diagram, showing the number on shapes where it is currently hidden and hiding the number on shapes where it is showing.

```
' Dimension the variables
Dim igxDiagramObject As DiagramObject
Dim igxShape As Shape
Dim igxShape1 As Shape
Dim igxShape2 As Shape
' Create two shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
igxShape2.ShapeNumber.Shown = True
MsgBox "Click OK to toggle the ShapeNumber state of each shape."
' Go through each DiagramObject in the Diagram
For Each igxDiagramObject In ActiveDiagram.DiagramObjects
   ' Is it a shape?
   If igxDiagramObject.Type = ixObjectShape Then
       Set igxShape = igxDiagramObject.Shape
       ' Toggle the shown state
       igxShape.ShapeNumber.Shown = Not igxShape.ShapeNumber.Shown
   End If
Next igxDiagramObject
MsgBox "Click OK to continue."
```

See Also Shape.ShowNumbering property

{button ShapeNumber object,JI(`igrafxrf.HLP',`ShapeNumber_Object')}

Trim Method

Syntax *ShapeNumber*.**Trim** (*NumParts* As Integer)

Description The Trim method sets all of the numbers beyond the supplied number of parts index to zero. For example, if the shape number is 10-23-4-7 and you call this method with a value of 2 for the *NumParts* argument, then the shape number would change to 10-23-0-0.

The *NumParts* argument specifies the number of parts in the shape number, starting from the left, to maintain their current values. All other number parts are reset to zero.

Example The following example creates a shape on the active diagram and sets up its shape number. It then modifies the second part of the shape number by changing it to a 2 and outputs the formatted value to the Immediate window. The shape number is then trimmed and the new shape number is displayed in a message box.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeNumber As ShapeNumber
Dim igxField As Field
Dim igxFieldText As FieldText
Dim igxNumberFormat As NumberFormat
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeNumber object form the shape
Set igxShapeNumber = igxShape.ShapeNumber
' Turn on shape numbers for the shape
igxShapeNumber.Shown = True
' Get the Field object
Set igxField = igxShapeNumber.Field
' Get the FieldText object
Set igxFieldText = igxField.FieldText
' Get the NumberFormat object
Set igxNumberFormat = igxFieldText.NumberFormat
' Set the format for the shape number. i.e.(X*X*X*X)
igxNumberFormat.NumberOfParts = 4
igxNumberFormat.Separator(1) = "*"
igxNumberFormat.Separator(2) = "*"
igxNumberFormat.Separator(3) = "*"
igxNumberFormat.IncrementingPart = 4
' Set the shape number's position to be below the shape
igxField.FieldPosition = ixFieldBelow
' Randomize the values of the number parts
For Index = 1 To igxNumberFormat.NumberOfParts
   igxShapeNumber.Part(Index) = Rnd(1) * 100
Next Index
' Output the old formatted value to a message box
MsgBox "The old formatted value is " & _
    igxShapeNumber.FormattedValue
' Trim the shape number to the first part
igxShapeNumber.Trim (1)
' Display the new formatted value
MsgBox "The values after trimming are " & _
    igxShapeNumber.FormattedValue
```

{button ShapeNumber object,JI(`igrafxrf.HLP',`ShapeNumber_Object')}

Value Property

Syntax *ShapeNumber*.**Value**

Data Type Long (read/write)

Description The Value property specifies the unformatted value of a shape number. This property only affects the first part of a shape number. When using single-part shape numbers, this is the number itself. However, when using multi-part shape numbers, the Value property is the first ppart of the number.

For example, a shape number that is set to have three parts, and is formatted may look like this 27-1-1. The Value property returns 27, even if the Part property is set to 3, for which in the example the value is 1.

Note that you can use the Part property to access and set the value of any particular part of the shape number. Therefore, using the example shape number given above, if you wanted to find out or change the value of the third part of the number, you would use the Part property with its *Index* argument set to 3.

The Value property is the ShapeNumber object's default property, allowing easy access to this information.

Example The following example creates a shape and sets a four part formatted number for the shape. It then displays the formatted value. What this example shows is that the Value property only refers to the first part of the shape number. If you are using multi-part numbers, you must access them through the Part property.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxShapeNumber As ShapeNumber
Dim igxField As Field
Dim igxFieldText As FieldText
Dim igxNumberFormat As NumberFormat
' Create the shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ShapeNumber object form the shape
Set igxShapeNumber = igxShape.ShapeNumber
' Turn on shape numbers for the shape
igxShapeNumber.Shown = True
' Get the Field object
Set igxField = igxShapeNumber.Field
' Get the FieldText object
Set igxFieldText = igxField.FieldText
' Get the NumberFormat object
Set igxNumberFormat = igxFieldText.NumberFormat
' Set the format for the shape number. i.e.(X*X*X*X)
igxNumberFormat.NumberOfParts = 4
igxNumberFormat.Separator(1) = "*"
igxNumberFormat.Separator(2) = "*"
igxNumberFormat.Separator(3) = "*"
' Set the shape number's position to be below the shape
igxField.FieldPosition = ixFieldBelow
' Change the Part property, and show that the Value property
' always returns the first part of the number
For iCount = 1 To 4
    PartNum = igxShapeNumber.Part(iCount)
    ' Show the Value property
```

```
    MsgBox "The Value of Part " & iCount & " is " _
        & igxShapeNumber.Value
Next iCount
MsgBox "The Value property always returns the " _
    & "first part of the number." & Chr(13) & "Click " _
    & "OK to try using the Value property to set the " _
    & Chr(13) & "value of a part other than the first part."
' Try to use the Value property to change the value of a Part
' other than the first part
For iCount = 1 To 4
    PartNum = igxShapeNumber.Part(iCount)
    ' Change the Value property
    igxShapeNumber.Value = iCount * 3
    MsgBox "The Value of Part " & iCount & " is " _
        & igxShapeNumber.Value
Next iCount
```

See Also

FormattedValue property

Part property

{button ShapeNumber object,JI(`igrafxrf.HLP',`ShapeNumber_Object')}

Document Object

The Document object is a container (a disk file) for storing any number of diagrams and components. It is subordinate to the Application object, and can be accessed through the Application object's Documents collection or ActiveDocument property.

A Document object contains any of the following objects:

- A Diagrams collection, which contains one or more Diagram objects
- A Components collection, which contains zero or more Component objects
- A DataFieldTemplates collection, which contains zero or more DataFieldTemplate objects
- A PropertyLists collection, which contains zero or more PropertyList objects

A Document object is also associated with:

- A CommandBars collection, which contains zero or more CommandBar objects
- A ShapeLibrary object, which contains zero or more ShapeLibraryItem objects
- A Windows collection, which contains zero or more Window objects

The Components dialog (File—Components) functions as a view of the components and diagrams contained in a document. For more information about components, refer to the Components collection object and the Component object.

The following example gets the ActiveDocument object from the Application object.

```
' Dimension the variables
Dim igxDocument As Document
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
```

Additionally, a Document object can be accessed from the Documents collection. The following example gets the first Document object from the Documents collection.

```
' Dimension the variables
Dim igxDocument As Document
' Get the Documents collection from the Application object
Set igxDocuments = Application.Documents.Item(1)
```

Properties, Methods, and Events

All of the properties, methods, and events for the Document object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
ActiveDiagram	ActivateDocument	Activate
ActiveView	CloseDocument	AfterPrint
AnyControls	DoAfterCurrentChangeBracket	BeforeClose
Application	DoAfterTopChangeBracket	BeforePrint
AsType	FireUserEvent	BeforeSave
CommandBars	MakeComponentRange	Close
Components	MakeDiagramRange	Deactivate
CustomDataDefinitions	OpenChangeBracket	FunctionValue

DefaultDiagramType
DefaultFormats
DepartmentNames
Diagrams
DiagramTypes
Entities
FullName
HasDiskFile
InPlaceActive
Modified
Name
Parent
Path
PermanentDocument
PropertyLists
Protected
ReadOnly
Saved
VBAName
Views
Windows

Protect
Run
SaveAsWebPage
SaveDocument
SaveDocumentAs
SendMail
Stop
Unprotect
UpdateFields
UpdateShapes

GetInterface
Modify
New
Open
PropertyChange
Save
UserEvent

Activate Event

Syntax **Private Sub** *Document*_**Activate**()

Description The Activate event occurs when the specified document is activated (that is, it obtains the focus). As an example, this event can be useful if you have a user form that is associated with the document, and you want to make sure it is always visible when the document is active. You can write code in this event to always show a particular user form, or to perform any other desired action.

Example The following example shows a user form named MyForm when the document is activated. This example assumes that a user form called MyForm exists. The event subroutine must be placed in a Document project.

```
Private Sub Document_Activate()
    ' Activate the userform if it is not visible
    ' when the document is activated.
    If Not (MyForm.Visible) Then
        MyForm.Show Modeless
    End If
End Sub
```

See Also ActivateDocument method

Deactivate event

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

ActivateDocument Method

Syntax *Document*.**ActivateDocument**

Description The ActivateDocument method activates the specified document; that is, it gives the specified document the focus.

Example The following example retrieves the last document from the Documents collection, and then activates it.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxDocuments As Documents
' Get the Documents collection from the Application object
Set igxDocuments = Application.Documents
' Get the last document from the Documents collection
Set igxDocument = igxDocuments.Item(igxDocuments.Count)
' Activate the document
igxDocument.ActivateDocument
```

See Also Activate event

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

ActiveDiagram Property

Syntax *Document*.**ActiveDiagram**

Data Type Diagram object (read-only, See Object Properties)

Description The ActiveDiagram property returns the currently active Diagram object for the specified Document object. The currently active diagram is the diagram that currently has the focus and is receiving user input (although an automation user could be providing input to a diagram that isn't active).

Example The following example gets the ActiveDiagram object from the ActiveDocument object, and then draws a shape on the active diagram.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
' Set the igxDiagram variable to the ActiveDiagram object
Set igxDiagram = Application.ActiveDocument.ActiveDiagram
' Create a new shape with its center at one inch and then
' set it to the igxShape variable
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, igxDiagram.DiagramType.ShapeLibrary.Item(1))
```

See Also Diagram object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

ActiveView Property

Syntax *Document*.**ActiveView**

Data Type View object (read-only, See Object Properties)

Description The ActiveView property returns the currently active View object for the specified Document object. The currently active view is the view that currently has the focus and is receiving user input.

Example The following example gets the ActiveView from the ActiveDocument object. It then gets the DiagramView object from the View object, and uses it to change the zoom percentage to 200%. Finally, the view is centered on the newly created shape.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxView As View
Dim igxDiagramView As DiagramView
Dim igxDiagram As Diagram
Dim igxShape As Shape
' Set the igxDiagram variable to the ActiveDiagram object
Set igxDiagram = Application.ActiveDocument.ActiveDiagram
' Set the igxDiagram variable to the ActiveDiagram object
Set igxDocument = Application.ActiveDocument
' Get the View object from the ActiveDocument object
Set igxView = igxDocument.ActiveView
' Get the DiagramView object from the View object
Set igxDiagramView = igxView.DiagramView
' Set the zoom percentage to 200%
igxDiagramView.ZoomPercentage = 200
' Create a new shape with its center at one inch and then
' set it to the igxShape variable
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, igxDiagram.DiagramType.ShapeLibrary.Item(1))
' Set the view to be centered on the shape
igxDiagramView.ScrollToObject igxShape.DiagramObject, True
```

See Also View object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

AfterPrint Event

Syntax **Private Sub** *Document*_**AfterPrint**()

Description The AfterPrint event occurs after the specified document is printed (that is, after a “Print” command has been issued for the document). Custom code can be written within this event procedure to perform any desired actions, such as informing the user that the document was printed or outputting data to a print log.

The AfterPrint event can also be used in conjunction with the BeforePrint event to do things like adding a watermark or updating a time or date on the document. In the BeforePrint event you could add a custom, print-time only watermark and in the AfterPrint event you could remove that watermark so it doesn't get in the way of editing the document.

Example The following example outputs the date and time that a document was printed to the ‘Print Log’ output pane of the output window. If the ‘Print Log’ output pane does not exist, it is created. The event subroutine must be placed in a Document project.

```
Private Sub Document_AfterPrint()
    ' Dimension the variables
    Dim igxOutputPane As OutputPane
    Dim igxOutputPanes As OutputPanes
    Dim igxOutputWindow As OutputWindow
    ' Get the OutputWindow object
    Set igxOutputWindow = Application.OutputWindow
    ' Get the OutputPanes collection object
    Set igxOutputPanes = igxOutputWindow.OutputPanes
    ' Set up error trapping
    On Error GoTo CreatePane
    ' Try to get the Print Log output pane.
    Set igxOutputPane = igxOutputPanes.Item("Print Log")
    ' Go to ContinueOutput if valid pane is retrieved
    GoTo ContinueOutput

CreatePane:
    ' Create a new pane if it does not exist
    Set igxOutputPane = Application.OutputWindow.OutputPanes.Add _
       ("Print Log")

ContinueOutput:
    ' Add the output string to the output pane
    igxOutputPane.AddString "The document '" & ThisDocument.Name & _
       "' was printed on " & Date & " at " & Time & "."
    ' Make the output window visible
    igxOutputWindow.Visible = True
End Sub
```

See Also BeforePrint event

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

AnyControls Property

Syntax *Document*.**AnyControls**

Data Type AnyControls object (read-only, See Object Properties)

Description The AnyControls property returns an AnyControls object. The AnyControls object is used to provide access to the "Any" controls that are at the document level (refer to the AnyControls object).

These "Any" controls are always available to the VBA Document project, but other clients and VBA projects may need access to these controls also. This object gives other clients that capability.

Example The following example creates an event sink for the AnyDiagram object so that the client can listen to events for all the diagrams in a document. This code could be placed in any VBA project item in any of the VBA projects. Here, we place it in the VBA ShapeProject in the ShapeClass project item.

A `MyAnyDiagram` variable of type Diagram is declared using the `WithEvents` keyword, which allows this variable to listen to events. Once you declare the `MyAnyDiagram` variable, it will appear in the list of objects associated with that project item. If you select the AnyDiagram object from the drop down list, you can select event handlers for that object and you can write code for them.

To hookup the `MyAnyDiagram` variable to the AnyDiagram control for the document, you must execute the macro, declared here as `EstablishSink`. This will enable the `MyAnyDiagram` object to start listening to diagram events. We call `EstablishSink` when the shape class initializes in this example.

```
' Dimension a variable for the event sink
Public WithEvents MyAnyDiagram as Diagram

Public Sub EstablishSink()
   Set MyAnyDiagram = ActiveDocument.AnyControls.AnyDiagram
End Sub

Private Sub Diagram_Activate()
   MsgBox "Diagram " + Diagram.Name + " just activated."
End Sub

Public Sub ShapeClass_Initialize()
   EstablishSink
End Sub
```

See Also AnyControls object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

AsType Property

Syntax *Document*.**AsType**(*TypeName* As String) As Object

Data Type Object (read-only, See Object Properties)

Description The AsType property allows you to add your own properties and methods to a document object, extending the object model. The properties and methods can be organized into one or more document types, using unique type names.

The *TypeName* argument is a string that names the custom type. It can be any string you choose, but it must be unique within the environment. In an integrated environment, other programmers may be accessing the document, and using it's AsType property. To prevent conflicting type names, it is suggested that you use your company or department name, followed by a descriptive type name (for example, "MyCompanyFactory")

Use the following basic steps to implement a custom property or method for the Document object.

1. Use Document.AsType ("my type name").MyMethod in your code.
2. Create a new Class, and design properties and methods in the class.
3. Set up the GetInterface event to check the TypeName string passed to it. If it matches your type name, set the Interface parameter equal to your new class.

When you use Document.AsType(*TypeName*) in your code, you gain access to the properties and methods that you have defined in the new Class. The Document.AsType property automatically fires an event called GetInterface. The GetInterface event can have one or more AsType's defined, each one distinguished by a unique type name. Based on the type name, the GetInterface event redirects execution to your new Class by setting the Interface parameter. If the Interface parameter is set to your new Class, the Class properties and methods become exposed to the Document object.

Example The following example sets up a document type called "Factory". It displays a message that displays the "FactoryType" property of Factory. The FactoryType property is defined in the Class1 class. The Main() subroutine displays the message. The GetInterface event recognizes "Factory" as a type, and sets the Interface parameter to Class1, exposing Class1 properties to the Document object.

Put the following block of code into a new Class Module called Class1. To make the new class, RightClick on "ThisDocument" in the Visual Basic Project Explorer, and select Insert->Class Module.

```
' Class1
' FactoryType property (read only)
Public Property Get FactoryType() As String
    FactoryType = "Automobile Plant"
End Property
' EmployeeCount property (read only)
Public Property Get EmployeeCount() As String
   EmployeeCount = 2210
End Property
```

Put the rest of the code for this example (below) into the "ThisDocument" code window, and then run the Main() subroutine.

```
' Dimension a variable that hears document events
Private WithEvents igxDocument As Document
```

```
' Run this to test the event
Sub Main()
    ' Set the document variable
   Set igxDocument = ActiveDocument
   ' "Factory" is what the GetInterface event will look for.
   ' "FactoryType" is a property in our custom class
   MsgBox "The factory is an " & _
       igxDocument.AsType("Factory").FactoryType
End Sub

' The GetInterface event is fired whenever the AsType method is used
Private Sub igxDocument_GetInterface(ByVal TypeName As String, Interface As
Object)
    ' If the broadcast type name is "Factory", then set the interface
   If TypeName = "Factory" Then
       ' TypeName gets broadcast everywhere, so we need to check if
       ' someone else grabbed and set "Interface" first.
       ' If "Interface" is "Nothing" then it's free to Set to Class1
       If Interface Is Nothing Then
           ' Redirect the Document property to our Class1
           Set Interface = New Class1
       Else
           ' If someone else set Interface first, display this message
       MsgBox "Cannot set the interface. Someone else" & _
           " is using AsType Factory"
       End If
   End If
End Sub
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

BeforeClose Event

Syntax **Private Sub** *Document*_**BeforeClose**(*Cancel* As Boolean)

Description The BeforeClose event occurs before the specified document is closed. This event can be useful if you want the user to perform some action, or you want to inform the user of something before the document is closed. If the user does not perform the action, you can set the *Cancel* parameter to True to prevent the document from being closed.

Example The following example asks the user if they are sure they want to close the document. If the user says no, then the closing of the document is canceled. The event subroutine must be placed in a Document project.

```
Private Sub Document_BeforeClose(Cancel As Boolean)
   If MsgBox("Are you sure you want to close this document?",  vbYesNo) =
vbNo Then
       Cancel = True
   End If
 End Sub
```

See Also Close event

CloseDocument method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

BeforePrint Event

Syntax **Private Sub** *Document*_**BeforePrint**()

Description The BeforePrint event occurs before any diagram in the document is printed on the system printer. This event can be useful if you want the user to perform some action, or you want to inform the user of something before the document is printed.

With the iGrafx Professional API, diagrams are printed individually at the Diagram object level (Diagram.PrintDiagram method). Therefore, this event is fired once for each diagram that is printed.

Document objects do not have a Print method; however, from the user interface users can choose to print whole documents by choosing that option in the Print Dialog box. In this case, this event fires only once, even if multiple diagrams are printed.

Example The following example displays a message box informing the user that a diagram in the current document is about to be printed. The event subroutine must be placed in a Document project.

```
Private Sub Document_BeforePrint()
    MsgBox "The a diagram in this document is about to be printed."
End Sub
```

See Also AfterPrint event

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

BeforeSave Event

Syntax **Private Sub** *Document*_**BeforeSave**(*Cancel* As Boolean)

Description The BeforeSave event occurs before the specified document is saved, but after the Save command has been requested. Custom code can be written within this event procedure to perform any desired actions. You can prevent, or cancel, the Save operation by setting the *Cancel* parameter to True.

For example, If you are holding on to some information in variables that needs to be saved with the document, you could write that information out to the document in the BeforeSave event, perhaps writing it out to a property list associated with the document. Placing this code in the BeforeSave event would ensure that the document is up to date before any save occurs.

Example The following example stores the last time the document was saved to disk. Also, the user has the option to cancel the save if desired, using the *Cancel* parameter. The example assumes that all of the code is placed in a Document project.

There are two ways you can code this example:

1. Place all of the following code in a Document project (ThisDocument, for example).
2. Place the event subroutine in a Document project (it *must* be in a Document project), and the Main subroutine in a Diagram project. The LastTimeDocSaved variable can be defined in either module, as long as it is properly referenced. For example, if it is defined in the Document project, then from the Diagram project you would write: "ThisDocument.LastTimeDocSaved".

```
' Dimension a module variable to store date and time
Private LastTimeDocSaved As String

Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    ' Create two shapes
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 4, 1440)
    ' Set the diagram object variable
    Set igxDiagramObject = igxShape2.DiagramObject
    MsgBox "Click OK to save the document."
    ActiveDocument.SaveDocumentAs "c:\test.igx"
    ' Pause for the user
    MsgBox "The Document was last saved to disk " & LastTimeDocSaved
End Sub

Private Sub Document_BeforeSave(Cancel As Boolean)
    If MsgBox("Save the document?", vbYesNo) = vbNo Then
        ' Cancel the save
        Cancel = True
    Else
        ' Store the date and time the shape was saved
        LastTimeDocSaved = Now
    End If
End Sub
```

See Also Save event

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Close Event

Syntax **Private Sub** *Document*_**Close**()

Description The Close event occurs when the specified document is closed. Custom code can be written within this event procedure to perform any desired actions. This event can be useful if you want to clean up variables or temporary files before the document is closed.

Example The following example uses the Document_Open and Document_Close events. The Close event displays the total number of seconds the document was opened. It calculates the duration by comparing the system time when opened, with the system time when closed. The event subroutines must be placed in a Document project.

```
' Dimension a module variable to store time in seconds
Private TimeOpened As Long

' The Open event stores the current system time
Private Sub Document_Open()
    TimeOpened = Timer
End Sub

' The Close event displays the duration the doc was opened
Private Sub Document_Close()
    ' Dimension the variables
    Dim Duration As Long
    ' Calculate duration document has been open
    Duration = Timer - TimeOpened
    ' Display the result, and ask the user to confirm closing
    MsgBox "The document was opened for " & Int(Duration) _
        & " seconds."
End Sub
```

See Also BeforeClose event

CloseDocument method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

CloseDocument Method

Syntax *Document*.**CloseDocument**

Description The CloseDocument method closes the specified document.

Example The following example gets the ActiveDocument object from the Application object. It then uses the CloseDocument method to close the active document.

```
' Dimension the variables
Dim igxDocument As Document
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Close the active document
igxDocument.Close
```

See Also Close event

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

CommandBars Property

Syntax *Document*.**CommandBars**

Data Type CommandBars collection (read-only, See Object Properties)

Description The CommandBars property returns the CommandBars collection for the specified Document object. The CommandBars collection can be used to add, delete, and manipulate the toolbars and menus of the iGrafx Professional application.

Changes you make to the toolbars and menus with the CommandBars object associated with a Document are in place for that document only. That is, if you add a menu using the CommandBars associated with Document1, then switch to Document2, the menu you added disappears. If you then switch back to Document1, the menu you added returns.

If you want to make changes to toolbars and menus that are in place for a particular diagram, use the CommandBars object associated with a Diagram object. If you want to make changes to toolbars and menus that affect all Documents, use the CommandBars object associated with the Application object.

Changes you make using the CommandBars object associated with a Document are not saved with the document. Therefore, it is best to make any document-related user interface changes in the document's Open event so that the changes are applied each time the document is opened.

Example The following example gets the CommandBars object from the ActiveDocument object. It then uses the CommandBars object to find the command bar named 'MyCommandBar'. A message box indicating success or failure is then displayed.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Get the command bars collection form the ActiveDocument object
Set igxCmdBars = igxDocument.CommandBars
' Set the results of the find to the igxCmdBar variable
Set igxCmdBar = igxCmdBars.Find("MyCommandBar")
' Display a message box indicating success or failure
If (igxCmdBar Is Nothing) Then
    MsgBox ("No matching command bar found.")
Else
    MsgBox ("Command bar found.")
End If
```

See Also CommandBar object

CommandBars object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Components Property

Syntax *Document*.**Components**

Data Type Components collection (read-only, See Object Properties)

Description The Components property returns the Components collection for the specified Document object. Components are objects that are contained in a document. iGrafx Professional includes Scenario and Report objects which can be added to a document as components. Other applications that use the iGrafx Professional system may include other components.

The Component dialog (File—Components) functions as a view of the components and diagrams contained in a document. It also allows you to create new components and diagrams in a document.

Example The following example uses the ActiveDocument object to get the Components collection. It then displays the count of the Components collection in a message box.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxComponents As Components
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Get the Components collection object
Set igxComponents = igxDocument.Components
' Display the count
MsgBox igxComponents.Count
```

See Also Component object

Components object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

CustomDataDefinitions Property

Syntax *Document*.**CustomDataDefinitions**

Data Type CustomDataDefinitions collection object (read-only, See Object Properties)

Description The CustomDataDefintions property returns the CustomDataDefinitions collection for the specified Document object. The CustomDataDefinitions collection contains the definitions of all the data fields associated with a document.

For example, if you have a document with Organizational charts in it; you may want to define some data fields to store an employee name, an employee id, and a current salary for each shape. After defining those three data fields, the document's CustomDataDefinitions collection would contain three definitions, one for an employee name field of type Text, one for an employee id of type Number, and one for a current salary of type Currency.

Example The following example gets the CustomDataDefinitions collection from the ActiveDocument object. It then uses the CustomDataDefinitions collection to add a new data field definition (of type Text) to the document.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxCustomDataDef As CustomDataDefinitions
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Get the CustomDataDefinitions collection object
Set igxCustomDataDef = igxDocument.CustomDataDefinitions
' Add a new text field to the CustomDataDefinitions collection
igxCustomDataDef.Add "Employee Name", ixCustomDataFormatTextBase
```

See Also CustomDataDefinition object

CustomDataDefinitions object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Deactivate Event

Syntax **Private Sub** *Document*_**Deactivate**()

Description The Deactivate event occurs when the specified Document is deactivated. This event is the opposite of the Activate event. This can be useful if you have a user form that is associated with a document, and you want to make sure it is hidden when the document is deactivated.

Example The following code hides a form associated with the document when the document is deactivated. This example assumes that a global variable called MyForm, which represents a user form, has already been declared. The event subroutine must be placed in a Document project.

```
Private Sub Document_Deactivate()
    ' Hide the user form when the document is deactivated
    MyForm.Hide
End Sub
```

See Also Activate event

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

DefaultDiagramType Property

Syntax *Document*.**DefaultDiagramType**

Data Type DiagramType object (read-only, See Object Properties)

Description The DefaultDiagramType property returns the default DiagramType object for the specified Document object. The default diagram type for iGrafx Professional is “Basic Diagram”; for iGrafx Process the default is a “Process” diagram.

Note The value of this property cannot be changed.

Example The following example uses the DefaultDiagramType property to find out what type of diagram is the default for the active document. It then gets the other diagram type, and adds a diagram of that type to the document.

```
Private Sub Diagram_New()
    ' Dimension the variables
   Dim igxDiagType As DiagramType
   ' Get the current default diagram type
   Set igxDiagType = ActiveDocument.DefaultDiagramType
   MsgBox "Default diagram type currently set to " _
    & igxDiagType.SingularName
    If (igxDiagType.SingularName = "Basic Diagram") Then
        ' Set the default to Process
        Set igxDiagType = Application.DiagramTypes.Item(1)
        MsgBox "Diagram type changed to " _
            & igxDiagType.SingularName
    End If
    ActiveDocument.Diagrams.AddOfType "My Process", igxDiagType
End Sub
```

See Also DiagramType object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

DefaultFormats Property

Syntax *Document*.**DefaultFormats**

Data Type DefaultFormats object (read-only, See Object Properties)

Description The DefaultFormats property returns the DefaultFormats object for the specified Document. The DefaultFormats object allows you to set up default formatting for shapes, connector lines, and source and destination arrows for the entire document.

Example The following example sets up default formats for shapes, connector lines, and arrows using the DefaultFormats object. It then adds two shapes and a connector line to demonstrate the result.

```
' Dimension the variables
Dim igxDocDefaults As DefaultFormats
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine As ConnectorLine
' Get the default formats object
Set igxDocDefaults = ActiveDocument.DefaultFormats
' Set the default shape, connector line, and arrow formats
With igxDocDefaults
    .ConnectorLineFillFormat.FillType = ixFillSolid
   .ConnectorLineFillFormat.FillColor = vbBlue
   .ConnectorLineLineFormat.Color = vbYellow
   .ConnectorLineLineFormat.Style = ixLineNormal
   .ConnectorLineLineFormat.Width = 40  ' Units are twips
   .ConnectorLineShadowFormat.Type = ixShadow1
   .ConnectorLineShadowFormat.Color = vbRed
   .ConnectorLineShadowFormat.Depth = 2
   ' Don't set connector 3D formats
   .SourceArrowFormat.Style = ixArrow10
   .SourceArrowFormat.Size = 2
   .SourceArrowFormat.Color = RGB(200, 150, 50)
   .DestinationArrowFormat.Style = ixArrow20
   .DestinationArrowFormat.Size = 3
   .DestinationArrowFormat.Color = RGB(50, 150, 200)
   .ShapeFillFormat.FillType = ixFillPattern
   .ShapeFillFormat.BackColor = vbCyan
   .ShapeFillFormat.FillColor = vbGreen
   .ShapeFillFormat.PatternIndex = 5
   .ShapeLineFormat.Style = ixLineDashed
   .ShapeLineFormat.Width = 60  ' Units are twips
   .ShapeLineFormat.Color = RGB(220, 40, 220)
   ' Don't set any shadow formatting for shapes
   .ShapeThreeDFormat.Type = ixThreeD12
   .ShapeThreeDFormat.Depth = 3
End With
' Pause for the user
MsgBox "Defaults set. Click OK to add two shapes and a connector."
' Add the shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
' Add the connector
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
```

```
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, ixDirSouth, _
    ixConnectRelativeToShape, , , igxShape2, ixDirNorth, _
    ixConnectRelativeToShape)
' Pause for the user
MsgBox "Click OK to continue."
```

See Also

DefaultFormats object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

DepartmentNames Property

Syntax *Document*.**DepartmentNames**

Data Type DepartmentNames collection object (read-only, See Object Properties)

Description The DepartmentNames property returns the DepartmentNames collection object for the specified Document. The DepartmentNames object allows you to determine what department names exist, and how many departments there are in a particular document.

Example The following example gets the DepartmentNames collection from the Document object. It then goes through the collection and outputs each department name in the collection in a message box.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxDeptNames As DepartmentNames
Dim strDeptName As String
Dim iCount As Integer
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Get the DepartmentNames object
Set igxDeptNames = igxDocument.DepartmentNames
' Output all of the names in the DepartmentNames
' collection in a message box
For iCount = 1 To igxDeptNames.Count
   MsgBox igxDeptNames.Item(iCount)
Next iCount
```

See Also DepartmentNames object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Diagrams Property

Syntax *Document*.**Diagrams**

Data Type Diagrams collection (read-only, See Object Properties)

Description The Diagrams property returns a Diagrams collection object for the specified Document object. The Diagrams collection can be used to access the individual diagrams that exist within the document.

Example The following example gets the first diagram from the Diagrams collection of the ActiveDocument object. It then activates the first diagram.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxDiagram As Diagram
Dim igxDiagrams As Diagrams
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Get the DiagramTypes collection
Set igxDiagrams = igxDocument.Diagrams
' Get the first diagram from the Diagrams collection
Set igxDiagram = igxDiagrams.Item(1)
' Activate the first diagram
igxDiagram.ActivateDiagram
```

See Also Diagram object

Diagrams object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

DiagramTypes Property

Syntax *Document*.**DiagramTypes**

Data Type DiagramTypes collection (read-only, See Object Properties)

Description The DiagramTypes property returns the DiagramTypes collection object for the specified Document object. The DiagramTypes collection can be used to access the individual diagram types that are available to the document.

Example The following example gets the DiagramTypes collection from the ActiveDocument object. It then goes through the collection and displays the template name of the diagram type, if one exists.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxDiagramTypes As DiagramTypes
Dim iCount As Integer
Dim strTemplateName As String
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Get the DiagramTypes collection
Set igxDiagramTypes = igxDocument.DiagramTypes
' Go through the DiagramTypes collection
For iCount = 1 To igxDiagramTypes.Count
    ' Get the name of the template
    strTemplateName = igxDiagramTypes.Item(iCount).TemplateName
    ' Display the correct text in a message box
    If strTemplateName = "" Then
        MsgBox "No template name available."
    Else
        MsgBox strTemplateName
    End If
Next iCount
```

See Also DiagramTypes object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

DoAfterCurrentChangeBracket Method

Syntax *Document*.**DoAfterCurrentChangeBracket**(*Callback* As Callback)

Description The DoAfterCurrentChangeBracket method directs the program to execute a method, but not until the current ChangeBracket is finished.

The method can do anything the programmer chooses, but it must reside in a Class, and the method must be called `Callback_Execute`. The Class must use `Implement Callback` (see the example.)

The *Callback* argument is a Callback object, derived from the Class containing the method you want to execute.

Example The following example sets up a Callback Class, and executes it using the DoAfterCurrentChangeBracket method. The Main( ) program has a loop that creates 50 shapes in the diagram. When this procedure finishes, a message is displayed.

The code below is the Callback Class. It is used as a way to notify the user when a ChangeBracket procedure has finished by displaying a message box. Put this block of code into a new Class. Create a new Class by selecting Insert->Class Module in the Visual Basic editor.

```
' Class1 - Will be used to NotifyWhenFinished
' Need to specify that this class uses callback
' This will make Class1 As Callback
Implements Callback
' "Execute" procedure is required in callback classes
' This one just displays a message
Private Sub Callback_Execute()
   MsgBox "The current ChangeBracket has finished."
End Sub
```

The following code is the main program. Put this block of code into the "ThisDocument" project, and run the routine.

```
Private Sub Main()
    ' Dimension the variables
   Dim igxDiagram As Diagram
   Dim igxShape As Shape
   Dim igxChangeBracket As ChangeBracket
   Dim index As Integer
   ' Set our new callback variable
   Set NotifyWhenFinished = New Class1
   ' Set the igxDiagram variable to the ActiveDiagram object
   Set igxDiagram = Application.ActiveDiagram
   ' Start a ChangeBracket
   Set igxChangeBracket = _
       Application.ActiveDocument.OpenChangeBracket("MyChangeBracket")
   ' Post the callback variable for executing after the current
   ' ChangeBracket is finished
   ActiveDocument.DoAfterCurrentChangeBracket(NotifyWhenFinished)
   ' Create 50 new shapes
   MsgBox "Click OK to add 50 new shapes."
   For index = 1 To 50
       Set igxShape = igxDiagram.DiagramObjects.AddShape _
           (100 * index, 100 * index, _
           igxDiagram.DiagramType.ShapeLibrary.Item(1))
```

```
    Next index
    igxChangeBracket.Close
End Sub
```

See Also

DoAfterTopChangeBracket method

ChangeBracket object

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

DoAfterTopChangeBracket Method

Syntax *Document*.**DoAfterTopChangeBracket**(*Callback* As Callback)

Description The DoAfterTopChangeBracket method

The DoAfterTopChangeBracket method directs the program to execute a method, but not until the top ChangeBracket is finished. The top bracket is the oldest existing bracket that has not yet been closed (Change Brackets can be nested).

The method can do anything the programmer chooses, but it must reside in a Class, and the method must be called `Callback_Execute`. The Class must use `Implement Callback` (see the example.)

The *Callback* argument is a Callback object, derived from the Class containing the method you want to execute.

Example The following example sets up a Callback Class, and executes it using the DoAfterCurrentChangeBracket method. The Main( ) program has a loop that creates 50 shapes in the diagram. When this procedure finishes, a message is displayed.

The code below is the Callback Class. It is used as a way to notify the user when a ChangeBracket procedure has finished by displaying a message box. Put this block of code into a new Class. Create a new Class by selecting Insert->Class Module in the Visual Basic editor.

```
' Make this a callback class
Implements Callback
' Callback classes have one required method called "Execute"
Private Sub Callback_Execute()
    ' Display a message
   MsgBox "The Top Change Bracket has been closed."
End Sub
```

The following code is the main program. Put this block of code into the "ThisDocument" code window, and run the routine.

```
Private Sub Main()
    ' Dimension the variables
   Dim igxDiagram As Diagram
   Dim igxChangeBracket1 As ChangeBracket
   Dim igxChangeBracket2 As ChangeBracket
   Dim index As Integer
   ' Set our new callback variable
   Set NotifyWhenFinished = New Class1
   ' Set the igxDiagram variable to the ActiveDiagram object
   Set igxDiagram = Application.ActiveDiagram
   ' Start two ChangeBrackets
   Set igxChangeBracket1 = _
       Application.ActiveDocument.OpenChangeBracket("BracketA")
   Set igxChangeBracket2 = _
       Application.ActiveDocument.OpenChangeBracket("BracketB")
   ' Post the callback variable for executing after the current
   ' ChangeBracket is finished
   ActiveDocument.DoAfterTopChangeBracket (NotifyWhenFinished)
   ' Create 25 new shapes
   MsgBox "Click OK to add 25 new shapes."
   For index = 1 To 25
       igxDiagram.DiagramObjects.AddShape _
```

```
            100 * index, 100 * index, _
            igxDiagram.DiagramType.ShapeLibrary.Item(1)
    Next index
    ' Close the first ChangeBracket
    igxChangeBracket1.Close
    MsgBox "Click OK to add 25 more shapes."
    ' Add 25 more shapes
    For index = 26 To 50
        igxDiagram.DiagramObjects.AddShape _
            100 * index, 100 * index, _
            igxDiagram.DiagramType.ShapeLibrary.Item(1)
    Next index
    ' Close the second ChangeBracket
    igxChangeBracket2.Close
    MsgBox "Click OK to continue."
End Sub
```

See Also DoAfterCurrentChangeBracket method

ChangeBracket object

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Entities Property

Syntax *Document*.**Entities**

Data Type Entities collection (read-only, See Object Properties)

Description The Entities property returns the Entities collection object for the specified Document object. The Entities collection can be used to access all of the entities that have been created in a document.

Example The following example gets the Entities collection from the ActiveDocument object. It then uses the Entities collection object to determine the name of the Shape object in which every entity resides, and display the shape's object name in a message box.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine As ConnectorLine
Dim igxEntity As Entity
Dim igxEntities As Entities
' Create 2 shapes in the diagram and connect them
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 3, 1440)
igxShape1.Text = "Shape 1"
igxShape2.Text = "Shape 2"
' Add connector line
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
   (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
   ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
   ixDirWest, ixConnectRelativeToShape)
' Create an entity in the first shape
Set igxEntity = ActiveDocument.Entities.Add("Entity1", igxShape1)
' Add a third shape and connect it to shape 2
Set igxShape1 = igxShape2
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 3, 1440 * 3)
igxShape2.Text = "Shape 3"
' Add connector line
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
   (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
   ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
   ixDirNorth, ixConnectRelativeToShape)
' Add a third shape and connect it to shape 2
Set igxShape1 = igxShape2
' Add an entity to the third shape
Set igxEntity = ActiveDocument.Entities.Add("Entity2", igxShape1)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440, 1440 * 3)
igxShape2.Text = "Shape 4"
' Add connector line
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
   (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
   ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
   ixDirEast, ixConnectRelativeToShape)
' Get the Entities collection
```

```
Set igxEntities = ActiveDocument.Entities
' Display the number of entities
MsgBox "The Entities collection contains " & igxEntities.Count _
   & " entities."
' Go through the Entities collection
For iCount = 1 To igxEntities.Count
   ' Get the Shape object
   Set igxShape1 = igxEntities.Item(iCount).Location
   ' Display the name of the object
   MsgBox "Item " & iCount & " in the Entities collection is in " _
       & igxShape1.Text
Next iCount
```

See Also

Entities object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

FireUserEvent Method

Syntax *Document*.**FireUserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The FireUserEvent method fires the "UserEvent" for the specified document. You can use this functionality to send messages to any document that is listening to document events.

You must specify an *EventIdentifier* argument (a string) to use for your event. You might choose to use something like your company name followed by the event name. You should choose a name that won't conflict with names picked by other developers.

You can pass one parameter to the event (the *Parameter* argument). This parameter is a Variant, so one logical choice is to pass a Class.

Then, you can write code in a UserEvent handler to perform some actions when your event fires. This code should be of the form:

```
If EventIdentifier = "<<Your identifier string>>" Then
   << Write your code here >>
End If
```

Example The following example defines a new user event called "ShowUsers". The *Parameter* argument that gets passed is a class, which has one property called Count. The event handler displays the passed parameter's Count property.

The following code defines a simple class with one property. Create a new class below a Diagram project called Class1, and copy this code into it.

```
' Class1
' It contains one property, read only
Public Property Get Count() As Long
   Count = 25
End Property
```

The following code is the main program. Copy this, and the UserEvent subroutine, into a Diagram project code window.

```
' Run this subroutine to test the event
Public Sub Main()
    ' Create a new Class1 object
   Dim MyClass1 As New Class1
   ' Fire the UserEvent
   ActiveDocument.FireUserEvent "ShowUsers", MyClass1
End Sub

' This event handler runs every time any FireUserEvent method
' is used in the system
Private Sub Document_UserEvent(ByVal EventIdentifier As String, ByVal
Parameter As Variant)
    ' Check if the Identifier string is the one we want
   If EventIdentifier = "ShowUsers" Then
       ' Redirect to Class1
       MsgBox "The number of users is " & Parameter.Count
   End If
End Sub
```

See Also UserEvent event

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

FunctionValue Event (iGrafx Process Only)

Syntax **Private Sub** *Document*_**FunctionValue**(*FunctionName* As String, *DoubleArgument* As Double, *DoubleResult* As Double)

Description The FunctionValue event provides a way to create process simulation functions that are handled by Visual Basic, rather than by the iGrafx Process built-in function facilities. The FunctionValue event occurs when a process simulation encounters a Visual Basic Model Function. Model Funtions are created from the iGrafx Process user interface from the Model->Functions… menu item. Model Functions are handled by Visual Basic if the "Visual Basic" check box is checked when creating the function.

The *FunctionName* parameter is passed to the event as the name of the Model Function defined in the process diagram. The FunctionValue event could potentially contain many separate routines, each handling a different function. Have your code look at this parameter value to determine which function to handle.

The *DoubleArgument* argument is the value passed to the event from the Model Function. (Double refers to double precision decimal values, which allow for accurate floating point math in the function.)

The *DoubleResult* argument is the answer that gets sent back to the function in the process diagram. Your code should fill in this parameter with the result of the function.

Example The following example uses the FunctionValue event to handle converting inches and centimeters. It checks which function is requested and evalutates the result. The event subroutine must be placed in a Document project.

```
Private Sub ThisDocument_FunctionValue(FunctionName As String, _
DoubleArgument As Double, DoubleResult As Double)
    ' Check the function name
   If FunctionName = "ConvertInchesToCentimeters" Then
       ' Convert inches to centimeters
       DoubleResult = DoubleArgument * 0.4
       ' Now that it's done, exit the event
       Exit Function
   End If
   ' Check the function name
   If FunctionName = "ConvertCentimetersToInches" Then
       ' Convert centimeters to inches
       DoubleResult = DoubleArgument * 2.5
       ' Now that it's done, exit the event
       Exit Function
   End If
End Sub
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

GetInterface Event

Syntax **Private Sub** *Document*_**GetInterface**(ByVal *TypeName* As String, *Interface* As Object)

Description The GetInterface event occurs when the Document.AsType property is used. The AsType property allows you to add your own properties and methods to a document object, extending the object model. The properties and methods can be organized into one or more document types, using unique type names.

The *TypeName* argument is a string that distinguishes the custom type. It can be any string the programmer chooses, but it must be unique within the environment. In an integrated environment, other programmers may be accessing the document, and using it's AsType property. To prevent conflicting type names, it is suggested that you use your company or department name, followed by a descriptive type name (for example, "MyCompanyFactory").

Use the following basic steps to implement a custom property or method for the Document object.

1. Use Document.AsType ("my type name").MyMethod in your code.
2. Create a new Class, and design properties and methods in the class.
3. Set up the GetInterface event to check the TypeName string passed to it. If it matches your type name, set the Interface parameter equal to your new class.

When you use Document.AsType(*TypeName*) in your code, you gain access to the properties and methods that you have defined in the new Class. The Document.AsType property automatically fires an event called GetInterface. The GetInterface event can have one or more AsType's defined, each one distinguished by a unique type name. Based on the type name, the GetInterface event redirects execution to your new Class by setting the Interface parameter. If the Interface parameter is set to your new Class, the Class properties and methods become exposed to the Document object.

Notes When you extend an iGrafx Professional object using the GetInterface event, you need to keep in mind that other developers may be using this event also. To be a good citizen, you should do the following:

- Be sure to pick a name that is likely to be unique for your AsType name. In the example above, "MyType" is too generic and it is possible that another developer could use the same name. Instead, follow the convention of using your name or your company name, a period, and a description of the type. For example, if you were writing a type that extended Application to add additional internet capabilities, and your company name was "Micrografx", you could name your AsType name "Micrografx.InternetExtension".
- When you write code in the GetInterface event, keep it simple. You should not do any time consuming operation in the GetInterface event such as querying a database or displaying a dialog box.
- When you write code in the GetInterface event, be aware of the current state of the Interface parameter. In the example above, this is illustrated by the code fragment "Interface Is Nothing". If this code fragment evaluates to true, then it is safe to Set the interface to your class. If this code fragment evaluates to false then someone else has already responded to the event and set the interface to their class. If this condition arises, you should try changing your AsType name.

Example The following example sets up a document type called "Factory". It displays a message that displays the "FactoryType" property of Factory. The FactoryType property is defined in the Class1 class. The Main() subroutine displays the message. The GetInterface event recognizes "Factory" as a type, and sets the Interface parameter to Class1, exposing Class1 properties to the Document object.

Put the following block of code into a new Class Module called Class1. To make the new class, RightClick on "ThisDocument" in the Visual Basic Project Explorer, and select Insert->Class Module.

```
' Class1
' FactoryType property (read only)
Public Property Get FactoryType() As String
    FactoryType = "Automobile Plant"
End Property
' EmployeeCount property (read only)
Public Property Get EmployeeCount() As String
   EmployeeCount = 2210
End Property
```

Put the rest of the code for this example (below) into the "ThisDocument" project, and then run the Main() subroutine.

```
' Dimension a variable that hears document events
Private WithEvents igxDocument As Document

' Run this to test the event
Sub Main()
    ' Set the document variable
   Set igxDocument = ActiveDocument
   ' "Factory" is what the GetInterface event will look for.
   ' "FactoryType" is a property in our custom class
   MsgBox "The factory is an " & _
        igxDocument.AsType("Factory").FactoryType
End Sub

' The GetInterface event is fired whenever the AsType method is used
Private Sub igxDocument_GetInterface(ByVal TypeName As String, Interface As
Object)
    ' If the broadcast type name is "Factory", then set the interface
   If TypeName = "Factory" Then
        ' TypeName gets broadcast everywhere, so we need to check if
        ' someone else grabbed and set "Interface" first.
        ' If "Interface" is "Nothing" then it's free to Set to Class1
        If Interface Is Nothing Then
            ' Redirect the Document property to our Class1
            Set Interface = New Class1
        Else
            ' If someone else set Interface first, display this message
        MsgBox "Cannot set the interface. Someone else" & _
            " is using AsType Factory"
        End If
   End If
End Sub
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

HasDiskFile Property

Syntax *Document*.**HasDiskFile**[= {**True | False**}]

Data Type Boolean (read-only)

Description The HasDiskFile property indicates whether the specified document has a file saved on disk. As mentioned in the description of the Document object, a document is just a disk file that contains some number of diagrams and components. If a document has been saved at least one time, then a disk file exists (unless it has been deleted), and the property returns True. If a document has not been saved, moved, or deleted from its originally opened location, the property's value is False.

When saving a document, the HasDiskFile indicates whether the SaveDocument or SaveDocumentAs method should be used. If the property returns True, either method can be used. If the property returns False, the SaveDocumentAs method must be used to save the document. If this property is False, then trying to use the SaveDocument method produces an error.

Example The following example gets the ActiveDocument object from the Application object. It then saves the document using Save, or SaveAs, depending whether the document already has a file on disk.

```
' Dimension the variables
Dim igxDocument As Document
' Get the ActiveDocument object from the Application object
Set igxDocument = Application.ActiveDocument
' Display a message based on the HasDiskFile property
MsgBox "Click OK to save the document."
' Use Save or SaveAs, depending on HadDiskFile
If igxDocument.HasDiskFile Then
    igxDocument.SaveDocument
Else
    igxDocument.SaveDocumentAs "c:\test.igx"
End If
```

See Also SaveDocument method

SaveDocumentAs method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

InPlaceActive Property

Syntax *Document*.**InPlaceActive**[= {**True | False**}]

Data Type Boolean (read-only)

Description iGrafx Professional documents can be placed in other documents, such as Word and Excel, as Ole objects. The InPlaceActive property indicates whether the specified document is currently in an OLE edit session inside another document. This information is important because some of the menus and tools that are available during an InPlace session are different than those when working in the iGrafx Professional editor. Also you may want to lock layers, or protect other aspects of diagrams when the document is in an InPlaceActive session.

Example The following example displays a message indicating whether the document is in an InPlace edit session. Try putting this code in the "ThisDocument" code window, and save the document. Then load the document as an object into Word, Excel, etc… and activate the iGrafx Professional document for editing.

```
Private Sub Test()
    If ThisDocument.InPlaceActive Then
        MsgBox "This document is an Object in another document," _
            & Chr(13) & "and it is in edit mode."
    Else
        MsgBox "This document is in the iGrafx Professional editor."
    End If
End Sub
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

MakeComponentRange Method

Syntax *Document*.**MakeComponentRange** As ComponentRange

Description The MakeComponentRange method creates a blank ComponentRange colelction object. The blank component range can then be used to create a custom component range using existing component objects.

Example The following example creates a ComponentRange, and adds all the components of the document (if any) to the ComponentRange collection. It then displays a PercentGauge as it proceeds.

```
' Dimension the variables
Dim igxComponentRange As ComponentRange
Dim Index As Integer
' Make the component range object
Set igxComponentRange = ThisDocument.MakeComponentRange
' Display a percent gauge
Application.PercentGauge.Visible = True
' Iterate through all components, add each to the range
For Index = 1 To ActiveDocument.Components.Count
    igxComponentRange.Add ActiveDocument.Components.Item(Index)
    ' Update the percent gauge
    Application.PercentGauge.Text = Index & "components added so far."
    Application.PercentGauge.Value = _
        (Index * 100) / ActiveDocument.Components.Count
    Application.PercentGauge.Visible = True
Next Index
' Remove the percent gauge
Application.PercentGauge.Visible = False
```

See Also ComponentRange object

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

MakeDiagramRange Method

Syntax *Document*. **MakeDiagramRange** As DiagramRange

Description The MakeDiagramRange method creates an empty DiagramRange collection object. The empty diagram range can then be used to create a custom diagram range using existing Diagram objects.

Example The following example creates a DiagramRange object, and three Diagram objects. It then adds the diagrams to the range, and iterates through the range to display the name of each diagram.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxDiagram3 As Diagram
Dim igxDiagramRange As DiagramRange
Dim sString As String
' Add three diagrams to the document
Set igxDiagram1 = ActiveDocument.Diagrams.Add("Diagram A")
Set igxDiagram2 = ActiveDocument.Diagrams.Add("Diagram B")
Set igxDiagram3 = ActiveDocument.Diagrams.Add("Diagram C")
' Make a DiagramRange
Set igxDiagramRange = ActiveDocument.MakeDiagramRange
' Add the diagrams to the range
igxDiagramRange.Add igxDiagram1
igxDiagramRange.Add igxDiagram2
igxDiagramRange.Add igxDiagram3
' Collect the names of the diagrams in the range
For Each Diagram In igxDiagramRange
   sString = sString & Diagram.Name & Chr(13)
Next Diagram
' Display the result
MsgBox "The Document contains these diagrams:" & _
   Chr(13) & Chr(13) & sString
```

See Also DiagramRange object

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Modified Property

Syntax *Document*.**Modified**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Modified property specifies whether the document has been modified. You can use this property to flag a document as modified or not. When this property is set to True, the Modify event of the Document object is fired.

Example The following example gets the ActiveDocument from the Application object. It then sets the Modified property to True, which activates the Modify event of the Document object. When the Modify event is fired, a message is displayed.

```
Private Sub Document_Modify()
    MsgBox "The document was modified."
End Sub

Public Sub MyTest()
   ' Dimension the variables
   Dim igxDocument As Document
   ' Get the ActiveDocument object from the Application object.
   Set igxDocument = Application.ActiveDocument
   ' Set the Modified property to True
   igxDocument.Modified = True
End Sub
```

See Also Modify event

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Modify Event

Syntax **Private Sub** *Document*_**Modify**()

Description The Modify event occurs when the Modified property is set to True or the diagram is modified either programmatically or by user intervention. This event is useful if there are a set of functions you need to run whenever a document is modified.

Example The following example gets the ActiveDocument from the Application object. It then sets the Modified property to True, which activates the Modify event of the Document object. When the Modify event is fired, a message is displayed. The event subroutine must be placed in a Document project.

```
Private Sub Document_Modify()
    MsgBox "The document was modified."
End Sub

Public Sub MyTest()
   ' Dimension the variables
   Dim igxDocument As Document
   ' Get the ActiveDocument object from the Application object
   Set igxDocument = Application.ActiveDocument
   ' Set the Modified property to True
   igxDocument.Modified = True
End Sub
```

See Also Modified property

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

New Event

Syntax **Private Sub** *Document*_**New**()

Description The New event occurs when a new document is created. Custom code can be written within this event procedure to perform any desired actions. This event is useful when you want to initialize something when a new document is created.

Example The following example adds a Date property to any new document. The Test( ) routine displays the Date property's value for the newest document. The event subroutine must be placed in a Document project.

```
' Dimension module level variables
Private NewestDocument As Document
Private igxProperty As Property

' This event adds a Date property to any new document
Private Sub AnyDocument_New()
    ' Dimension variable
   Dim igxPropertyList As PropertyList
   ' Add a PropertyList to the document
   Set igxPropertyList = AnyDocument.PropertyLists.Add("Creation")
   ' Add a Property to the document
   Set igxProperty = igxPropertyList.Add("Date")
   ' Set the value to the creation date
   igxProperty.Value = Now
   ' Grab the current document object for later use
   Set NewestDocument = AnyDocument.PermanentDocument
End Sub

' Run this subroutine to view the property
Private Sub Test()
    ' Display the creation date property
   MsgBox NewestDocument.Name & " was created " & _
       igxProperty.Value
End Sub
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Open Event

Syntax **Private Sub** *Document*_**Open**()

Description The Open event occurs when a document is opened. Custom code can be written within this event procedure to perform any desired actions. This event is useful when you want to create or initialize something when a document is opened.

Example The following example uses the UpdateShapes method in the Open event for the document to update the shapes in a document every time the document is opened. The event subroutine must be placed in a Document project.

```
Private Sub Document_Open()
    ' Dimension the variables
    Dim igxDocument As Document
    ' Get the ActiveDocument object
    Set igxDocument = Application.ActiveDocument
    ' Update the shapes in the document
    igxDocument.UpdateShapes
End Sub
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

OpenChangeBracket Method

Syntax *Document*.**OpenChangeBracket** (*bstrName* As String) As ChangeBracket

Description The OpenChangeBracket method returns a ChangeBracket object. A ChangeBracket object is used to organize, or group, a series of commands so they can be easily undone by a user. A change bracket is also used to organize a series of commands so they can be processed together quickly.

The *bstrName* argument is a string value that specifies the name that appears in the Undo area of the Edit menu.

The OpenChangeBracket method works closely with the ChangeBracket.Close method. After using OpenChangeBracket, any subsequent commands are stored in the ChangeBracket, similar to recording a macro. To stop adding commands to a ChangeBracket, use the ChangeBracket.Close method.

When you open a ChangeBracket, the ChangeBracket appears in the iGrafx Professional Edit menu as an Undo option. The user can click it to perform the undo, which will undo all actions performed while the ChangeBracket was open.

Example The following example opens a change bracket named “Drawing Shapes”. It then draws nine shapes in the diagram, and then closes the change bracket.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxDiagram As Diagram
Dim igxDiagramObjects As DiagramObjects
Dim igxChangeBracket As ChangeBracket
' Get the ActiveDocument object from the Application object
Set igxDocument = Application.ActiveDocument
' Get the ChangeBracket object
Set igxChangeBracket = igxDocument.OpenChangeBracket("Drawing Shapes")
' Get the Diagram object
Set igxDiagram = igxDocument.ActiveDiagram
' Get the DiagramObjects object
Set igxDiagramObjects = igxDiagram.DiagramObjects
' Draw nine shapes in the diagram
igxDiagramObjects.AddShape 1440, 1440
igxDiagramObjects.AddShape 1440 * 2, 1440
igxDiagramObjects.AddShape 1440 * 3, 1440
igxDiagramObjects.AddShape 1440, 1440 * 2
igxDiagramObjects.AddShape 1440 * 2, 1440 * 2
igxDiagramObjects.AddShape 1440 * 3, 1440 * 2
igxDiagramObjects.AddShape 1440, 1440 * 3
igxDiagramObjects.AddShape 1440 * 2, 1440 * 3
igxDiagramObjects.AddShape 1440 * 3, 1440 * 3
' Close the change bracket
igxChangeBracket.Close
```

See Also ChangeBracket object

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

PermanentDocument Property

Syntax *Document*.**PermanentDocument**

Data Type Document object (read-only, See Object Properties)

Description The PermanentDocument property returns a Document object. The purpose of this property is to provide a means of holding on to the object an AnyControl is pointing at after an event is over; in this case, a Document object.

The AnyControl objects are special VBA controls that are only valid during an event; these objects dynamically point at the "active" object that is triggering the event. The PermanentDocument property is used to “grab” the specific object the AnyControl is pointing at so that it can be used (or accessed) once the event is over.

As an example, consider the following event procedure written for the AnyDocument_PropertyChange event.

```
Private Sub AnyDocument_PropertyChange()
    Set MyDocument = AnyDocument
End Sub
```

If the variable MyDocument is a global variable of type Document, then within the PropertyChange event you can set MyDocument to the Document object that is currently active. However, if you try to use MyDocument after the event is over, it returns an error because an event is not in progress. Since you set MyDocument to the AnyControl, your variable is pointing at the AnyControl that is dynamically pointing at the active object, which is Nothing outside of an event.

If your intent is to hold on to the specific document that the AnyDocument control is pointing at inside the event, then you need to use the PermanentDocument property. This property gives you a Document object that is valid after the event is over (outside of the event). The change to your code is as follows (MyDocument is a global variable of type Document):

```
Private Sub AnyDocument_PropertyChange()
    Set MyDocument = AnyDocument.PermanentDocument
End Sub
```

Example The following example adds a Date property to any new document. The Test( ) routine displays the Date property of the newest document. The PermanentDocument property sets the NewestDocument variable during the New event.

```
' Dimension module level variables
Private NewestDocument As Document
Private igxProperty As Property

' This event adds a Date property to any new document
Private Sub AnyDocument_New()
    ' Dimension the variables
   Dim igxPropertyList As PropertyList
   ' Add a PropertyList to the document
   Set igxPropertyList = AnyDocument.PropertyLists.Add("Creation")
   ' Add a Property to the document
   Set igxProperty = igxPropertyList.Add("Date")
   ' Set the value to the creation date
   igxProperty.Value = Now
   ' Grab the current document object for later use
   Set NewestDocument = AnyDocument.PermanentDocument
End Sub
```

```
' Run this subroutine to view the property
Private Sub Test()
    ' Display the creation date property
   MsgBox NewestDocument.Name & " was created " & _
       igxProperty.Value
End Sub
```

See Also

PermanentConnectorLine property

PermanentDepartment property

PermanentDiagram property

PermanentDiagramObject property

PermanentShape property

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

PropertyChange Event

Syntax **Private Sub** *Document*_**PropertyChange**(*Property* As Property)

Description The PropertyChange event occurs when a property of the specified document is changed. Property objects are added to document PropertyList objects, which are in PropertyLists collections.

Example The following example adds a Date property to the document. It then changes the value to the current date, which fires the event. The event displays the result of the change. The event subroutine must be placed in a Document project.

```
Private Sub Text()
    ' Dimension the variables
   Dim igxPropertyList As PropertyList
   Dim igxProperty As Property
   ' Add a PropertyList to the document
   Set igxPropertyList = ActiveDocument.PropertyLists.Add("Creation")
   ' Add a Property to the document
   Set igxProperty = igxPropertyList.Add("Date")
   ' Set the value to the creation date
   igxProperty.Value = Now
End Sub

Private Sub AnyDocument_PropertyChange(ByVal Property As IGrafx3.IXProperty)
   MsgBox "The " & Property.Name & " property was changed to: " _
       & Property.Value
End Sub
```

See Also PropertyLists property

Property object

PropertyList object

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

PropertyLists Property

Syntax *Document*.**PropertyLists**

Data Type PropertyLists collection object (read-only, See Object Properties)

Description The PropertyLists property returns the PropertyLists collection for the specified Document object. A PropertyLists collection allows you to create, delete, access, and manipulate the individual property lists of a document. Property lists are data structures that allow you to store variant values within a document.

Example The following example gets the PropertyLists collection object from the ActiveDocument object. It then uses the PropertyLists collection object to add a new property list to the document, if it does not already exist.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxPropertyLists As PropertyLists
Dim igxPropertyList As PropertyList
Dim fFound As Boolean
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Get the PropertyLists object
Set igxPropertyLists = igxDocument.PropertyLists
' Check to see if the property list already exists
Found = igxPropertyLists.ItemExists("MyPropertyList")
' If the property list was not found, then add it
If Not Found Then
    ' Add a property list named MyPropertyList
    Set igxPropertyList = igxPropertyLists.Add("MyPropertyList")
Else
    ' Display a message box if it already exists
    MsgBox "Property List already exists."
    ' Get property list
    Set igxPropertyList = igxPropertyLists.Item("MyPropertyList")
End If
```

See Also PropertyList object

PropertyLists object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Protect Method

Syntax *Document*.**Protect** (*Password* As String)

Description The Protect method associates a password with a document and locks it, preventing the document from being modified. The *Password* argument can be any character sequence considered to be valid for the String data type in Visual Basic. The protection can be removed from a document by using the Unprotect method. You can also use the Protected property to determine whether a document is protected or unprotected. The maximum number of characters for a password is eight. All characters past the eighth are ignored.

Example The following example tests whether a document is protected and if it is then it uses the unprotect method to turn off the protection. If it is not protected, then it uses the protect method to protect the document.

```
' Dimension the variable
Dim igxDocument As Document
' Get the ActiveDocument obejct
Set igxDocument = Application.ActiveDocument
' See if the document is protected
If igxDocument.Protected Then
    ' Unprotect the document
    igxDocument.Unprotect "Password"
Else
    ' Protect the document
    igxDocument.Protect "Password"
End If
```

See Also Protected property

Unprotect method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Protected Property

Syntax *Document*.**Protected**[= {**True | False**}]

Data Type Boolean (read-only)

Description The Protected property indicates whether the document is password protected. A document can be protected with the Protect method. It can be unprotected by using the Unprotect method.

Example The following example tests whether a document is protected and if it is then it uses the unprotect method to turn off the protection. If it is not protected, then it uses the protect method to protect the document.

```
' Dimension the variable
Dim igxDocument As Document
' Get the ActiveDocument obejct
Set igxDocument = Application.ActiveDocument
' See if the document is protected
If igxDocument.Protected Then
    ' Unprotect the document
    igxDocument.Unprotect "Password"
Else
    ' Protect the document
    igxDocument.Protect "Password"
End If
```

See Also Protect method

Unprotect method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

ReadOnly Property

Syntax *Document*.**ReadOnly**[= {**True | False**}]

Data Type Boolean (read-only)

Description The ReadOnly property indicates whether the document is read-only, or can be saved. If the specified document is read-only, changes made to the document cannot be saved. If the document is read-only, the effect is the same as when the read-only attribute is set for a file within the File Manager or OS Explorer.

Example The following example reads the ReadOnly property in the Modify event to post a message box indicating whether any changes to the document can be saved. Use this example code on a document that has, and has not, had its read-only flag (through the file system) set.

```
' Dimension a variable that hears document events
Private WithEvents igxDocument As Document

Private Sub Test()
   ' Dimension the variables
   Dim igxDiagType As DiagramType
   ' Get the current default diagram type
   Set igxDiagType = ActiveDocument.DefaultDiagramType
   MsgBox "Default diagram type currently set to " _
       & igxDiagType.SingularName
    If (igxDiagType.SingularName = "Basic Diagram") Then
       ' Set the default to Process
       Set igxDiagType = Application.DiagramTypes.Item(1)
    End If
    ActiveDocument.Diagrams.AddOfType "My Process", igxDiagType
End Sub

Private Sub igxDocument_Modify()
    ' Test if document is read only
    If igxDocument.ReadOnly Then
       ' Display a message box indicating that document is read only
       MsgBox "Changes made to the document will " _
            & "not be saved because the document is read only."
    Else
        MsgBox "Document is not Read-Only--Changes are allowed"
    End If
End Sub
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Run Method

Syntax *Document*.**Run**

Description The Run method begins the execution of the entities, iDiagrams, associated with a document. All of the entities begin and continue execution simultaneously. This method runs all of the entities associated with a document. To run a specific entity, use the Entity.Run method.

Example The following example gets the ActiveDocument object from the Application object and then runs the entities using the Run method.

```
' Dimension the variables
Dim igxDocument As Document
' Get the ActiveDocument obejct
Set igxDocument = Application.ActiveDocument
' Begin the entities executing
igxDocument.Run
```

See Also Stop method

Entity.Run method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Save Event

Syntax **Private Sub** *Document*_**Save**()

Description The Save event occurs when a document is saved (refer to the SaveDocument or SaveDocumentAs methods). Custom code can be written within this event procedure to perform any desired actions. This event can be useful if you want to update a database or other data source when a file is saved.

Example The following example stores the date and time whenever the document is saved. The date is then retreived and displayed in a message. The event subroutine must be placed in a Document project.

```
' Dimension a module variable to store date and time
Private LastTimeDocSaved As String

Private Sub Main()
    ' Dimension variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Create two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440 * 3)
   ' Set the diagram object variable
   Set igxDiagramObject = igxShape2.DiagramObject
   MsgBox "Click OK to save the document."
   ActiveDocument.SaveDocumentAs "c:\test.igx"
   ' Pause for the user
   MsgBox "The Document was last saved to disk " & LastTimeDocSaved
End Sub

Private Sub Document_Save()
    If MsgBox("Save the document?", vbYesNo) = vbNo Then
       ' Cancel the save
       MsgBox "You clicked No. Document will not be saved."
    Else
        ' Store the date and time the document was saved
        LastTimeDocSaved = Now
    End If
End Sub
```

See Also Saved property

SaveDocument method

SaveDocumentAs method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

SaveAsWebPage Method

Syntax *Document*.**SaveAsWebPage**(*Diagrams* As DiagramRange, [*Components* As ComponentRange], *Folder* As String, [*DiagramsAsJava* As Boolean = False], [*OutputNotes* As Boolean = True], [*OutputLinkedDocuments* As Boolean = False], [*DiagramZoomPercent* As Integer = 100]) As String

Description The SaveAsWebPage method saves the currently open iGrafx Professional document as an HTML file. The method returns a string. The return value is the path and file name of the resulting HTML file.

The *Diagrams* argument specifies the diagrams to use in creating the HTML file. The order of the diagrams in the DiagramRange collection is important if you expect to get the proper results.

The *Components* argument is optional; it specifies the Component objects to use in constructing the HTML file. The argument's type is ComponentRange, and the order of components in the collection is an important consideration.

The *Folder* argument specifies the name of a file system folder. This folder is where the HTML file and its related files are stored. This argument is required.

The *DiagramsAsJava* argument specifies whether the web page uses Java code to display the diagrams. If set to True, the diagrams are saved as Java applets, along with HTML pages which display the diagrams. If set to False, the web page is saved without using Java applets. This argument is optional; the default is False.

The *OutputNotes* argument specifies whether shape notes are included in the HTML file. If set to True, shape notes are included. If set to False, shape notes are excluded. This argument is optional; the default is True.

The *OutputLinkedDocuments* argument specifies whether any documents that are linked to the current document are saved in the HTML file. If set to True, linked iGrafx Professional documents are included. If set to False, they are not included. This argument is optional; the default is False

The *DiagramZoomPercentage* argument provides a way to adjust the size of the graphic elements that appear in the web page. A value of 100 causes objects in the web page to appear at the original size. Values greater than 100 make the graphic objects larger. Values less than 100 make the graphic objects smaller. This argument is optional; the default is 100.

Example The following example saves a document as a web page. It sets up the document with two diagrams. It puts a shape on one diagram which includes a note and a link. The web page is saved using the Java, OutputNotes, and OutputLinkedDocuments, and the zoom is enlarged to 110%.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxDiagRange As DiagramRange
Dim strWebPage As String
' Get the active diagram object
Set igxDiagram1 = ActiveDiagram
' Add a new diagram
Set igxDiagram2 = ThisDocument.Diagrams.Add("Diagram X")
' Add a shape to diagram 1
Set igxShape1 = igxDiagram1.DiagramObjects.AddShape(1440, 1440)
' Add a link to the shape
igxShape1.Links.AddDiagramLink "Diagram X"
' Add a note to the shape
igxShape1.Note.Text = "This is a note"
```

```
' Make a DiagramRange
Set igxDiagRange = ThisDocument.MakeDiagramRange
' Add both diagrams to the range
igxDiagRange.Add igxDiagram1
igxDiagRange.Add igxDiagram2
' Save the document as a web page
strWebPage = ThisDocument.SaveAsWebPage _
    (igxDiagRange, , "c:\html", True, True, True, 110)
MsgBox "The page has been saved as " & strWebPage
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Saved Property

Syntax *Document*.**Saved**[= {**True | False**}]

Data Type Boolean (read-only)

Description The Saved property indicates whether any changes made to the document have been saved. When a document is opened (either a new or an existing document), initially this property is set to True, because no changes have occurred. Once a change has been made to the document, either programmatically or interactively, this property is set to False.

The property is read only, and is useful for determining whether a document needs to be saved. This property can be set to True by using either the SaveDocument or SaveDocumentAs methods.

Example The following example saves the document and displays the Saved property. Then the document is altered, and the Saved property is displayed again, to show the result.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxShape1 As Shape
' Get the active diagram object
Set igxDiagram1 = ActiveDiagram
' Add a shape to diagram 1
Set igxShape1 = igxDiagram1.DiagramObjects.AddShape(1440, 1440)
MsgBox "Click OK to save the document."
ThisDocument.SaveDocumentAs "c:\test.igx"
MsgBox "Document.Saved = " & ThisDocument.Saved & Chr(13) & _
    "Click OK to move the shape."
igxShape1.DiagramObject.CenterX = _
    igxShape1.DiagramObject.CenterX + 200
MsgBox "Document.Saved = " & ThisDocument.Saved
```

See Also HasDiskFile property

Save event

SaveDocument method

SaveDocumentAs method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

SaveDocument Method

Syntax *Document*.**SaveDocument**

Description The SaveDocument method saves the currently open document to disk. This method provides the same functionality as the File – Save menu option from the iGrafx Professional interface.

The SaveDocument method saves the currently open document to disk, using the path and file name of the previous save. This method provides the same functionality as the File–Save menu option from the user interface. This method can only be used if the document has already been saved at least once. If not, the method produces an error.

Example The following example gets the ActiveDocument object from the Application object. It then saves the document using Save, or SaveAs, depending whether the document already has a file on disk.

```
' Dimension the variables
Dim igxDocument As Document
' Get the ActiveDocument object from the Application object
Set igxDocument = Application.ActiveDocument
' Display a message based on the HasDiskFile property
MsgBox "Click OK to save the document."
' Use Save or SaveAs, depending on HadDiskFile
If igxDocument.HasDiskFile Then
    igxDocument.SaveDocument
Else
    igxDocument.SaveDocumentAs "c:\test.igx"
End If
```

See Also HasDiskFile property

Save event

Saved property

SaveDocumentAs method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

SaveDocumentAs Method

Syntax *Document*.**SaveDocumentAs**(*FileName* As String)

Description The SaveDocumentAs method saves a document to disk using the path and file name that is supplied for the *Filename* argument. This method provides the same functionality as the File–Save As menu option from the user interface.

Example The following example gets the ActiveDocument object from the Application object. It then saves the document using Save, or SaveAs, depending whether the document already has a file on disk.

```
' Dimension the variables
Dim igxDocument As Document
' Get the ActiveDocument object from the Application object
Set igxDocument = Application.ActiveDocument
' Display a message based on the HasDiskFile property
MsgBox "Click OK to save the document."
' Use Save or SaveAs, depending on HadDiskFile
If igxDocument.HasDiskFile Then
    igxDocument.SaveDocument
Else
    igxDocument.SaveDocumentAs "c:\test.igx"
End If
```

See Also HasDiskFile property

Save event

Saved property

SaveDocument method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

SendMail Method

Syntax *Document*.**SendMail**

Description The SendMail method opens a new, blank E-mail message (using your installed E-mail software) with the specified iGrafx Professional document already inserted as a file attachment. This method does not automatically send E-mail. The user must edit and send the E-mail message manually.

Example The following example opens a new E-mail message with the iGrafx Professional document already inserted as a file attachment. Place this code in the Document-level code module.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxShape1 As Shape
' Get the active diagram object
Set igxDiagram1 = ActiveDiagram
' Add a shape to diagram 1
Set igxShape1 = igxDiagram1.DiagramObjects.AddShape(1440, 1440)
MsgBox "Click OK to open your E-mail software."
ThisDocument.SendMail
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Stop Method

Syntax *Document*.**Stop**

Description The Stop method terminates the execution of all entities that are currently running in the document.

Example The following example gets the ActiveDocument object from the Application object and then stops all of the running entities using the Stop method.

```
' Dimension the variables
Dim igxDocument As Document
' Get the ActiveDocument obejct
Set igxDocument = Application.ActiveDocument
' Stop the entities executing
igxDocument.Stop
```

See Also Run method

Entity.Stop method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Unprotect Method

Syntax *Document*.**Unprotect**(*Password* As String)

Description The Unprotect method removes the protection from a protected document so it can be edited. The Password arument specifies the document's password (set by the Protect method). If the Password argument is invalid, then an error is returned. The maximum number of characters for a password is eight. All characters past the eighth are ignored.

Error IGRAFX_E_INVALIDPASSWORD

Example The following example tests whether a document is protected and if it is then it uses the unprotect method to turn off the protection.

```
' Dimension the variable
Dim igxDocument As Document
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' See if the document is protected
If igxDocument.Protected Then
    ' Unprotect the document
    igxDocument.Unprotect "Password"
End If
```

See Also Protect method

Protected property

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

UpdateFields Method

Topic Under Construction!!!

Syntax *Document*.**UpdateFields**

Description The UpdateFields method

Example The following example

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

UpdateShapes Method

Syntax *Document*.**UpdateShapes** As Integer

Description The UpdateShapes method updates all of the shapes in the ShapeLibraries of the diagrams within a document with the shapes in iGrafx Share. You can use this method to have changes made in one central location, iGrafx Share, be updated globally. This method is the same selecting Update Shapes from the Shape Library dialog.

Example The following example uses the UpdateShapes method in the Open event for the document to update the shapes in a document every time the document is opened.

```
Private Sub Document_Open()
    ' Dimension the variables
    Dim igxDocument As Document
    ' Get the ActiveDocument object
    Set igxDocument = Application.ActiveDocument
    ' Update the shapes in the document
    igxDocument.UpdateShapes
End Sub
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

UserEvent Event

Syntax **Private Sub** *Document*_**UserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The UserEvent event provides a means of implementing your own custom events. Your custom events can then be triggered with the FireUserEvent method, which fires the specified "UserEvent" on the document. You can use this functionality to send messages to any objects listening to document-level events.

You must pick an event identifier string to use for your event. You might choose to use something like your company name followed by the event name. You should choose a name that won't conflict with names picked by other developers.

You can pass one parameter to the event. This parameter is a Variant, so one logical choice is to pass a class.

You then write code in a UserEvent handler to perform some actions when your event fires. This code should be of the form:

```
If EventIdentifier = "<<Your identifier string>>" Then
   << Write your code here >>
End If
```

Example The following example defines a new user event called "ShowUsers". The Parameter that gets passed is a class, which has one property called Count. The event handler displays the passed parameter's Count property.

The following code implements a simple class with one property. Create a new class below a diagram project called Class1 and copy this code into it.

```
' Class1
' It contains one property, read only
Public Property Get Count() As Long
   Count = 25
End Property
```

The following code is the main program. Copy this, and the UserEvent subroutine, into the "ThisDocument" project code window.

```
' Run this subroutine to test the event
Public Sub Main()
   ' Create a new Class1 object
   Dim MyClass1 As New Class1
   ' Fire the UserEvent
   ThisDocument.FireUserEvent "ShowUsers", MyClass1
End Sub

' This event handler runs every time any FireUserEvent method
' is used in the system
Private Sub ThisDocument_UserEvent(ByVal EventIdentifier As String, ByVal
Parameter As Variant)
   ' Check if the Identifier string is the one we want
   If EventIdentifier = "ShowUsers" Then
      ' Redirect to Class1
      MsgBox "The number of users is " & Parameter.Count
   End If
End Sub
```

See Also FireUserEvent method

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

VBAName Property

Syntax *Document*.**VBAName**

Data Type String (read/write)

Description The VBAName property is a string value that specifies the programmatic name of a Document object. The string can only contain letters and numbers. No spaces, punctuation marks, or other special characters can be used in the VBAName property.

If you open a document and look at the Document Project in the Visual Basic editor, you see that the name of the Document object defaults to “ThisDocument”. This is the VBAName (the example code verifies this). As shown in the example, you can change this name to any name you want. Run the example code and observe the name of the document listed in the project window of the VB editor.

Example The following example displays the VBAName of a document in a message box.

```
' Dimension the variables
Dim igxDocument As Document
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Display the VBAName
MsgBox "The value of the document's VBAName property is: " _
    & igxDocument.VBAName
' Change the property's value
igxDocument.VBAName = "Document4"
MsgBox "The VBAName property was changed." & Chr(13) _
    & "The value of the document's VBAName property is: " _
    & igxDocument.VBAName
```

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Views Property

Syntax *Document*.**Views**

Data Type Views object (read-only, See Object Properties)

Description The Views property returns the Views collection of the specified Document object. The Views object provides access to the document's View objects.

Example The following example adds two new views to the document. It then iterates through the Views collection and activates each view.

```
' Put this example code in the "ThisDocument" project code window
' Add two diagram views
ThisDocument.Views.AddDiagramView ActiveDiagram
ThisDocument.Views.AddDiagramView ActiveDiagram
' Iterate through each view
Dim Index As Integer
For Index = 1 To ThisDocument.Views.Count
   MsgBox "Click OK to activate the next view."
   ' Activate the next view
   ThisDocument.Views.Item(Index).Window.Activate
Next Index
```

See Also Views object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Windows Property

Syntax *Document*.**Windows**

Data Type Windows collection object (read-only, See Object Properties)

Description The Windows property returns the Windows collection of the specified Document object. The Windows collection can be used to access and manipulate the windows associated with a document.

Example The following example activates the first window in the Windows collection.

```
' Dimension the variables
Dim igxWindows As Windows
Dim igxWindow As Window
Set igxWindows = ThisDocument.Windows
Set igxWindow = igxWindows.Item(1)
igxWindow.Activate
```

See Also Window object

Windows object

iGrafx API Object Hierarchy

{button Document object,JI(`igrafxrf.HLP',`Document_Object')}

Documents Object

The Documents object is a collection of individual Document objects that are currently open in the iGrafx Professional application. A Documents collection is associated with and accessible from the Application object. Its purpose is to store and provide access to the individual Document objects that have been opened or created.

This object provides the following functionality:

- The ability to create new blank documents.
- The ability to create new documents from templates.
- The ability to create new documents of a specific type.
- The ability to open previously saved documents.
- The ability to access any Document objects that has been opened or created.
- The ability to determine how many Document objects are currently in the collection.

The following example gets the Documents collection object from the application object.

```
' Dimension the variables
Dim igxDocuments As Documents
' Get the Documents object
Set igxDocuments = Application.Documents
```

Properties, Methods, and Events

All of the properties, methods, and events for the Documents object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count	New	
Parent	NewFromTemplate	
	NewOfType	
	Open	

Item Method

Syntax *Documents*.**Item**(*Index* As Integer) As Document

Description The Item method returns the Document object at the specified *Index* from the Documents collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Document. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example adds a new document to the Documents collection, and then displays the name of each document in the collection.

```
' Dimension the variables
Dim igxDocument As Document
Set igxDocument = Documents.New
MsgBox "New document created." & Chr(13) & _
    "Its name is: " & igxDocument.FullName
```

{button Documents object,JI(`igrafxrf.HLP',`Documents_Object')}

New Method

Syntax *Documents*.**New** As Document

Description The New method creates and returns a new blank document. If you need to create a new document based on a template, then use the NewFromTemplate method. The NewOfType method creates a new document based on a DiagramType object.

Example The following example gets the Documents collection object from the Application object. It then uses the Documents collection object to create a new blank document.

```
' Dimension the variables
Dim igxDocuments As Documents
Dim igxDocument As Document
' Get the Documents collection object
Set igxDocuments = Application.Documents
' Create a new blank document
Set igxDocument = igxDocuments.New
MsgBox "View the result"
```

See Also NewFromTemplate method

NewOfType method

Document_New event

{button Documents object,JI(`igrafxrf.HLP',`Documents_Object')}

NewFromTemplate Method

Syntax *Documents*.**NewFromTemplate**(*TemplateFileName* As String) As Document

Description The NewFromTemplate method creates a new document based on the Template that matches the TemplateFileName argument. If you need to create a diagram from a DiagramType object, then use the NewOfType method. The New method creates and returns a blank document. If an invalid template name is supplied then an error is returned.

Error IGRAFX_E_INVALIDFILENAME

Example The following example attempts to create a new document based on a template that matches the supplied template name. If an invalid name is supplied, then the error is trapped and a message box is displayed and the value of nothing is returned. The Document object is returned if the file name supplied is valid. Run the Main() subroutine, which calls a user-defined function named “CreateNewFromTemplate”.

```
Public Sub Main()
    ' Dimension the variables
    Dim igxDocument As Document
    Set igxDocument = CreateNewFromTemplate _
        ("c:\Program Files\iGrafx\Pro\8.0\Template\Relation.igt")
End Sub

Function CreateNewFromTemplate(strFileName As String) As Document
   ' Dimension the variables
   Dim igxDocuments As Documents
   ' Get the Documents object
   Set igxDocuments = Application.Documents
   ' Create error trapping
   On Error GoTo ErrorTest
   ' Attempt to create a new document
   Set CreateNewFromTemplate = igxDocuments.NewFromTemplate _
       (strFileName)
   ' Exit the function if the document is created
    Exit Function

ErrorTest:
   ' Test the error returned.
   If Err = IGRAFX_E_INVALIDFILENAME Then
       ' Output message box
       MsgBox "The file name you supplied '" & _
           strFileName & "' does not exist."
   End If
   ' Set CreateNewFromTemplate to Nothing
   Set CreateNewFromTemplate = Nothing
End Function
```

See Also New method

NewOfType method

Document_New event

{button Documents object,JI(`igrafxrf.HLP',`Documents_Object')}

NewOfType Method

Syntax *Documents*.**NewOfType**(*DiagramType* As DiagramType) As Document

Description The NewOfType method creates a new document based on the DiagramType object argument. If you need to create a new document based on a template then use the NewFromTemplate method. The New method creates and returns a blank document.

Example The following example gets the first DiagramType object from the ActiveDocument. It then uses the DiagramType object to create a new document that is the same type as the specified DiagramType object.

```
' Dimension the variables
Dim igxActiveDocument As Document
Dim igxNewDocument As Document
Dim igxDocuments As Documents
Dim igxDiagramType As DiagramType
Dim igxDiagramTypes As DiagramTypes
' Get the Documents collection object
Set igxDocuments = Application.Documents
' Get the ActiveDocument object
Set igxDocument = Application.ActiveDocument
' Get the DiagramTypes collection object
Set igxDiagramTypes = igxDocument.DiagramTypes
' Get the first diagram type from the diagram types collection
Set igxDiagramType = igxDiagramTypes.Item(1)
' Create a new document that matches the type of the
' first diagram type of the active document
Set igxNewDocument = igxDocuments.NewOfType(igxDiagramType)
```

See Also New method

NewFromTemplate method

Document_New event

{button Documents object,JI(`igrafxrf.HLP',`Documents_Object')}

Open Method

Syntax *Documents*.**Open**(*FileName* As String) As Document

Description The Open method opens a previously saved document. If an invalid file name is supplied, then an error is returned.

Error IGRAFX_E_INVALIDFILENAME

Example The following example uses the Main() subroutine to call a user-defined function call “OpenFile”. The function uses the Open method to open a file that exists on disk somewhere. If an invalid name is supplied, then the error is trapped and a message box is displayed and the value of Nothing is returned. The Document object is returned if the file name supplied is valid. Change the file name in the Main() subroutine to a file that is valid on your system.

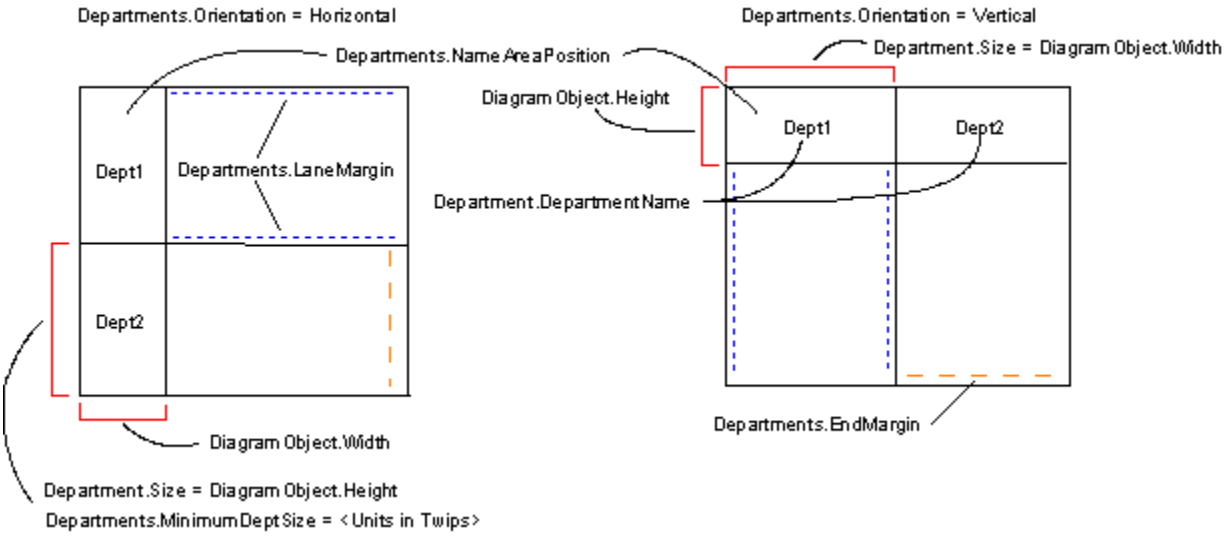
```
Public Sub Main()
    ' Dimension the variables
    Dim igxDocument As Document
    Set igxDocument = OpenFile("c:\My Documents\test.igx")
End Sub

Function OpenFile(strFileName As String) As Document
   ' Dimension the variables
   Dim igxDocuments As Documents
   ' Get the Documents object
   Set igxDocuments = Application.Documents
   ' Create error trapping
   On Error GoTo ErrorTest
   ' Attempt to open the document
   Set OpenFile = igxDocuments.Open(strFileName)
   ' Exit the function if the document is opened
   Exit Function
ErrorTest:
   ' Test the error returned
   If Err = IGRAFX_E_INVALIDFILENAME Then
       ' Output message box
       MsgBox "The file name you supplied '" & _
       strFileName & "' does not exist."
   End If
   ' Set Open file to Nothing
   Set OpenFile = Nothing
End Function
```

{button Documents object,JI(`igrafxrf.HLP',`Documents_Object')}

Department Object

The Department object represents the dividing of a diagram into groupings called departments. Dividing a diagram into departments is typical of process maps, where certain activities are performed by a specific department in an organization. However, the concept of departments in iGrafx Professional can be used for any type of diagram requiring that activities, actions, or events be associated with a specific group, organization, etc. For more information about the use of departments in iGrafx Professional, refer to the iGrafx Professional User's Guide.

Controlling the look of departments as drawn on diagrams is controlled through several API objects. The following diagram illustrates various properties associated with departments. Although not specifically called out, all of the blue dashed lines indicate the Departments.LaneMargin property, and all the orange dashed lines indicate the Departments.EndMargin property.



The following example shows how to get a department object from the Departments collection of the active diagram.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartments As Departments
Dim igxDepartment As Department
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection from the ActiveDiagram object
Set igxDepartments = igxDiagram.Departments
' Get the first department from the Departments collection
Set igxDepartment = igxDepartments.Item(1)
```

Notes

- All departments must be oriented in the same direction. Make sure that the Departments.Orientation is set correctly before adding departments to a diagram.
- The Size property controls the height when the orientation is horizontal, and the width when the orientation is vertical.
- The department size automatically increases to fit the text in a department's name area if the text requires more room than is allocated by the Size property.
- The DepartmentSize cannot be set to a value that is less than the value of the MinimumDeptSize property.

- Sizing the width of the department name area in horizontal orientation or the height in vertical orientation is done through properties of the DiagramObjects object. Several other properties of the DiagramObjects object can affect the look of a department.

Properties, Methods, and Events

All of the properties, methods, and events for the Department object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	Rename
BlockFormat	MoveDown	
DepartmentIndex	MoveUp	
DepartmentName		
DiagramObject		
FillFormat		
Paragraphs		
Parent		
PermanentDepartment		
ProcessFillFormat		
Size		
Text		
TextLF		
TextRange		

Related Topics

- Departments object
- DepartmentNames object
- DepartmentRange object
- iGrafx API Object Hierarchy

BlockFormat Property

Syntax *Department*.**BlockFormat**

Data Type BlockFormat object (read-only, See Object Properties)

Description The BlockFormat property returns the BlockFormat object associated with the specified Department object. This BlockFormat object controls the formatting of the text in the department's name area. Each Department object's name area has its own distinct BlockFormat object for controlling text formatting.

Example The following example creates three departments in the active diagram. It then gets the first department in the Departments collection and sets various properties of its BlockFormat object. Finally, it copies the first department's BlockFormat object to the other two departments.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartments As Departments
Dim igxDepartment As Department
Dim igxNextDept As Department
Dim igxBlockFmt As BlockFormat
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection from the ActiveDiagram object
Set igxDepartments = igxDiagram.Departments
' Add three departments to the diagram
igxDepartments.AddDepartment ("Accounting")
igxDepartments.AddDepartment ("Shipping")
igxDepartments.AddDepartment ("Manufacturing")
' Get the first department from the Departments collection
Set igxDepartment = igxDepartments.Item(1)
' Get the BlockFormat object from the Department object
Set igxBlockFmt = igxDepartment.BlockFormat
' Set the block formatting properties
igxBlockFmt.FillType = ixFillSolid
igxBlockFmt.FillColor = RGB(200, 200, 200)
igxBlockFmt.LineStyle = ixLineNormal
igxBlockFmt.LineColor = vbRed
igxBlockFmt.VerticalAlignment = ixVerticalAlignTop
igxBlockFmt.HorizontalAlignment = ixHorizontalAlignLeft
igxBlockFmt.Orientation = ixOrientation0
' Assign the block formatting from the first dept to
' the other two depts
Set igxNextDept = igxDepartments.Item(2)
Set igxNextDept.BlockFormat = igxDepartment.BlockFormat
Set igxNextDept = igxDepartments.Item(3)
Set igxNextDept.BlockFormat = igxDepartment.BlockFormat
```

See Also BlockFormat object

iGrafx API Object Hierarchy

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

DepartmentIndex Property

Syntax *Department*.**DepartmentIndex**

Data Type Long (read-only)

Description The DepartmentIndex property returns the index number of the specified Department object. The index number is the identifying number of the department in the Departments collection. For instance, the DepartmentIndex value of Departments.Item(1) is 1.

This property provides a simple way to determine a department's index number within the Departments collection when more than one department exists in a diagram. However, it is most useful when you are referring to a department by some means other than by number with the Item method.

Example The following example creates three departments in the active diagram, and then displays the department name and its index. The top and bottom departments are moved with the MoveUp and MoveDown methods, and then the department names and their indexes are displayed again.

```
' Dimension the variables
Dim igxDepartments As Departments
Dim igxDepartment As Department
Dim sString As String
' Get the Departments collection from the ActiveDiagram object
Set igxDepartments = ActiveDiagram.Departments
' Add three departments to the diagram
igxDepartments.AddDepartment ("Accounting")
igxDepartments.AddDepartment ("Shipping")
igxDepartments.AddDepartment ("Manufacturing")
' Display the department names and indexes
For Each igxDepartment In igxDepartments
   sString = sString & "Department: " & igxDepartment.DepartmentName _
       & ", Index = " & igxDepartment.DepartmentIndex & Chr(13)
Next
MsgBox "The diagram has the following departments: " _
   & Chr(13) & sString
' Move the first and last departments
igxDepartments.Item(1).MoveDown
igxDepartments.Item(3).MoveUp
' Reset the string and display the department names and indexes
sString = ""
For Each igxDepartment In igxDepartments
   sString = sString & "Department: " & igxDepartment.DepartmentName _
       & ", Index = " & igxDepartment.DepartmentIndex & Chr(13)
Next
MsgBox "The diagram has the following departments: " _
   & Chr(13) & sString
```

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

DepartmentName Property

Syntax *Department*.**DepartmentName**

Data Type String (read/write)

Description The DepartmentName property specifies the text that is displayed in the Name Area of a department. The department name can be any text string, and can contain carriage returns. The size of the department name area increases automatically to display the entire department name. Text formatting (font type, font size, etc.) is controlled with the BlockFormat property.

The text contained within the DepartmentName property is the same as that contained in the Text and TextLF properties. The main difference is that simulation methods use the DepartmentName property, and the object manipulation methods use the Text and TextLF properties.

Example The following example gets the ActiveDocument object and then creates a new department on the active diagram. It then changes the department name of the new department using the DepartmentName property.

```
' Dimension the variables
Dim igxDepartments As Departments
Dim igxDepartment As Department
Dim igxNextDept As Department
Dim igxBlockFmt As BlockFormat
' Get the Departments collection from the ActiveDiagram object
Set igxDepartments = ActiveDiagram.Departments
' Add three departments to the diagram
igxDepartments.AddDepartment ("Accounting")
igxDepartments.AddDepartment ("Shipping")
igxDepartments.AddDepartment _
    ("Manufacturing" & Chr$(13) & "Components Division")
MsgBox "View the diagram"
' Show that the DepartmentName, Text, and TextLF properties
' all contain the same string for each department
Set igxDepartment = igxDepartments.Item(1)
MsgBox "DepartmentName property: The first department " _
    & "name is " & igxDepartment.DepartmentName
MsgBox "Text property: The first department name is " _
    & igxDepartment.Text
MsgBox "TextLF property: The first department name is " _
    & igxDepartment.TextLF
Set igxDepartment = igxDepartments.Item(2)
MsgBox "DepartmentName property: The second department " _
    & "name is " & igxDepartment.DepartmentName
MsgBox "Text property: The second department name is " _
    & igxDepartment.Text
MsgBox "TextLF property: The second department name is " _
    & igxDepartment.TextLF
Set igxDepartment = igxDepartments.Item(3)
MsgBox "DepartmentName property: The third department " _
    & "name is " & igxDepartment.DepartmentName
MsgBox "Text property: The third department name is " _
    & igxDepartment.Text
MsgBox "TextLF property: The third department name is " _
    & igxDepartment.TextLF
' Change the Department name of the second department.
```

```
Set igxDepartment = igxDepartments.Item(2)
igxDepartment.DepartmentName = "Shipping" & Chr$(13) _
   & "Components Division"
MsgBox "View the diagram"
```

See Also

Text property

TextLF Property

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

DiagramObject Property

Syntax *Department*.**DiagramObject**

Data Type DiagramObject object (read-only, See Object Properties)

Description The DiagramObject property returns a DiagramObject object that is a Department object (Type equals Department). Some methods and properties of the DiagramObject object are not valid when the Type is Department. Refer to the documentation of the DiagramObject object for more information.

Example The following example creates a new department on the active diagram. It then uses the DiagramObject property of the department to set the Object name of the DiagramObject object equal to the Department.DepartmentName property.

```
' Dimension the variables
Dim igxDepartment As Department
Dim igxDiagramObject As DiagramObject
' Create a new department
Set igxDepartment = ActiveDiagram.Departments.AddDepartment _
    ("Department 1")
' Get the Diagram object
Set igxDiagramObject = igxDepartment.DiagramObject
' Set the ObjectName property
igxDiagramObject.ObjectName = igxDepartment.DepartmentName
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

FillFormat Property

Syntax *Department*.**FillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The FillFormat property returns a FillFormat object that is used to specify fill formatting for the name area of a department. The process area is controlled separately by the ProcessFillFormat property. The FillFormat object controls whether a fill is used, and if so, what type of fill (solid, pattern, or gradient), and the color or colors used.

Example The following example creates a department in the active diagram. Through the Department object, the process area's fill color is set to a solid yellow, and the name area's fill color is set to a solid blue.

```
' Dimension the variables
Dim igxDepartment As Department
Dim igxProcessFillFormat As FillFormat
Dim igxFillFormat As FillFormat
' Create a new department
Set igxDepartment = ActiveDiagram.Departments.AddDepartment _
    ("Department 1")
MsgBox "View the diagram"
' Get the ProcessFillFormat object and set properties
Set igxProcessFillFormat = igxDepartment.ProcessFillFormat
igxProcessFillFormat.FillType = ixFillSolid
igxProcessFillFormat.FillColor = vbYellow
MsgBox "View the diagram"
' Get the FillFormat object and set properties
Set igxFillFormat = igxDepartment.FillFormat
igxFillFormat.FillType = ixFillSolid
igxFillFormat.FillColor = vbBlue
MsgBox "View the diagram"
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

MoveDown Method

Syntax *Department*.**MoveDown** As Integer

Description The MoveDown method moves the specified department down a lane. For example, if a department named “Manufacturing” is the first department (topmost if the orientation is horizontal), and you want it to be the second department, apply the MoveDown method to the Manufacturing department. The department that was second becomes the first department. All other departments keep their previous order.

The method returns an integer result, and must be assigned to a variable of type Integer. The returned value represents the new index position of the department that was moved down. If the department is the last department, then an index that matches the index value of the last department is returned. This makes it easy to tell when a department is at the bottom of the other departments.

Example The following example consists of two subroutines. The Main() subroutine sets up three departments in a diagram. Then it calls the MoveDepartmentToBottom subroutine, which takes a Department object as an argument. It then uses the MoveDown method to move the department specified by the igxDepartment argument to the last position.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxDepartments As Departments
    ' Get the Departments collection from the ActiveDiagram object
    Set igxDepartments = ActiveDiagram.Departments
    ' Add three departments to the diagram
    igxDepartments.AddDepartment ("Accounting")
    igxDepartments.AddDepartment ("Shipping")
    igxDepartments.AddDepartment _
        ("Manufacturing" & Chr$(13) & "Components Division")
    MsgBox "Click OK to move the Accounting department to the bottom"
    MoveDepartmentToBottom igxDepartments.Item(1)
    MsgBox "View the result"
End Sub

Sub MoveDepartmentToBottom(igxDepartment As Department)
     ' Dimension the variables
     Dim intOldIndex As Integer
     Dim intCurrentIndex As Integer
     ' Set the old index value
     intOldIndex = igxDepartment.MoveDown
     ' Loop until the department is at the bottom
     While intOldIndex <> intCurrentIndex
         ' Set the old index value to the current index value.
         intOldIndex = intCurrentIndex
         ' Set current index value.
         intCurrentIndex = igxDepartment.MoveDown
     Wend
End Sub
```

See Also MoveUp method

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

MoveUp Method

Syntax *Department*.**MoveUp** As Integer

Description The MoveUp method moves the specified department up a lane. For example, if a department named “Manufacturing” is the second department, and you want it to be the first department, apply the MoveUp method to the Manufacturing department. The department that was first becomes the second department. All other departments keep their previous order.

The method returns an integer result, and must be assigned to a variable of type Integer. The returned value represents the new index position of the department that was moved up. If the department is at the top, then a value of one is returned every time the method is called. This makes it easy to tell when a department is at the top of the other departments.

Example The following example consists of two subroutines. The Main() subroutine sets up three departments in a diagram. Then it calls the MoveDepartmentToTop subroutine, which takes a Department object as an argument. It then uses the MoveUp method to move the department specified by the igxDepartment argument to the first position.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxDepartments As Departments
    ' Get the Departments collection from the ActiveDiagram object
    Set igxDepartments = ActiveDiagram.Departments
    ' Add three departments to the diagram
    igxDepartments.AddDepartment ("Accounting")
    igxDepartments.AddDepartment ("Shipping")
    igxDepartments.AddDepartment _
        ("Manufacturing" & Chr$(13) & "Components Division")
    MsgBox "Click OK to move the Accounting department to the bottom"
    MoveDepartmentToTop igxDepartments.Item(3)
    MsgBox "View the result"
End Sub

Sub MoveDepartmentToTop(igxDepartment As Department)
     ' Dimension the variables
     Dim intOldIndex As Integer
     Dim intCurrentIndex As Integer
     ' Set the old index value
     intOldIndex = igxDepartment.MoveUp
     ' Loop until the department is at the bottom
     While intOldIndex <> intCurrentIndex
         ' Set the old index value to the current index value
         intOldIndex = intCurrentIndex
         ' Set current index value
         intCurrentIndex = igxDepartment.MoveUp
     Wend
End Sub
```

See Also MoveDown method

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

Paragraphs Property

Syntax *Department*.**Paragraphs**

Data Type Paragraphs collection object (read-only, See Object Properties)

Description The Paragraphs property returns a Paragraphs collection object for the specified Department object. The Paragraphs object, through the Item method, provides access to the individual Paragraph objects. The Paragraphs collection for a Department object relates only to the name area.

Example The following example adds a department to the active diagram. It then sets the alignment of the first paragraph in the Paragraphs collection object to Left and sets the line spacing to 24 points. It then sets the alignment of the second paragraph to Center.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxParagraphs As Paragraphs
' Get the ActiveDiagram object.
Set igxDiagram = Application.ActiveDiagram
' Create a new department.
Set igxDepartment = igxDiagram.Departments.AddDepartment _
    ("Department 1" & Chr$(13) & "Components Div.")
MsgBox "View the state of the diagram"
' Get the paragraphs collection
Set igxParagraphs = igxDepartment.Paragraphs
' Set first paragraph alignment to Left and line spacing to 24 points
igxParagraphs.Item(1).ParagraphFormat.Alignment = _
    ixHorizontalAlignLeft
MsgBox "View the state of the diagram"
igxParagraphs.Item(1).ParagraphFormat.LineSpacingPoints = 24
MsgBox "View the state of the diagram"
' Set second paragraph alignment to Center
igxParagraphs.Item(2).ParagraphFormat.Alignment = _
    ixHorizontalAlignCenter
MsgBox "View the state of the diagram"
```

See Also Paragraph object

Paragraphs object

iGrafx API Object Hierarchy

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

PermanentDepartment Property

Syntax *Department*.**PermanentDepartment**

Data Type Department object (read-only, See Object Properties)

Description The PermanentDepartment property returns a Department object. The purpose of this property is to provide a means of holding on to the object an AnyControl is pointing at after an event is over; in this case, a Document object.

The AnyControl objects are special VBA controls that are only valid during an event; these objects dynamically point at the "active" object that is triggering the event. The PermanentDepartment property is used to “grab” the specific object the AnyControl is pointing at so that it can be used (or accessed) once the event is over.

As an example, consider the following event procedure written for the AnyDepartment_Rename event.

```
Private Sub AnyDepartment_Rename()
    Set MyDepartment = AnyDepartment
End Sub
```

If the variable MyDepartment is a global variable of type Department, then within the Rename event you can set MyDepartment to the Department object that is currently active. However, if you try to use MyDepartment after the event is over, it returns an error because an event is not in progress. Since you set MyDepartment to the AnyControl, your variable is pointing at the AnyControl that is dynamically pointing at the active object, which is Nothing outside of an event.

If your intent is to hold on to the specific Department object that the AnyDepartment control is pointing at inside the event, then you need to use the PermanentDepartment property. This property gives you a Department object that is valid after the event is over (outside of the event). The change to your code is as follows (MyDepartment is a global variable of type Department):

```
Private Sub AnyDepartment _Rename()
    Set MyDepartment = AnyDepartment.PermanentDepartment
End Sub
```

Example The following example uses the AnyDepartment_Modify event to set a variable to the Department object that originated the event. The Main( ) subroutine adds a new Department, and then renames it, which fires the Modify event. It then displays the name of the most recently modified department.

```
' Dimension a module variable
Private igxMostRecentlyModifiedDepartment As Department

Private Sub Main()
    ' Dimension a subroutine variable
   Dim igxDepartment As Department
   ' Add a department
   Set igxDepartment = ThisDocument.ActiveDiagram _
       .Departments.AddDepartment("Shipping")
   ' Rename the department
   igxDepartment.DepartmentName = "Receiving"
   ' Display the most recently modified department
   MsgBox "The most recently modified Department is " & _
       igxMostRecentlyModifiedDepartment.DepartmentName
End Sub
```

```
' An AnyDepartment event
Private Sub AnyDepartment_Modify()
    ' Get a permanent version of the Deparment
   ' that originated the event
   Set igxMostRecentlyModifiedDepartment = _
       AnyDepartment.PermanentDepartment
End Sub
```

See Also

PermanentConnectorLine property

PermanentDiagram property

PermanentDiagramObject property

PermanentDocument property

PermanentShape property

iGrafx API Object Hierarchy

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

ProcessFillFormat Property

Syntax *Department*.**ProcessFillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The ProcessFillFormat property returns a FillFormat object that is used to specify fill formatting for the process area of a department. The department name area is controlled separately by the FillFormat property. The ProcessFillFormat object controls whether a fill is used, and if so, what type of fill (solid, pattern, or gradient), and the color or colors used.

Example The following example creates a department in the active diagram. Through the Department object, the process area's fill color is set to yellow.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxProcessFillFormat As FillFormat
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Create a new department
Set igxDepartment = igxDiagram.Departments.AddDepartment _
    ("Department 1")
MsgBox "View the state of the diagram"
' Get the ProcessFillFormat object and set properties
Set igxProcessFillFormat = igxDepartment.ProcessFillFormat
igxProcessFillFormat.FillType = ixFillSolid
igxProcessFillFormat.FillColor = vbYellow
MsgBox "View the state of the diagram"
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

Rename Event

Syntax **Private Sub** *Department*_**Rename**(*Oldname* As String)

Description The Rename event occurs after the name of the specified Department object is changed. A name change occurs when either the DepartmentName property of the Department object is changed programmatically, or when a user edits the department name text through the user interface.

The *Oldname* parameter provides the previous name of the department (the name before it was changed).

Example The following example responds to the Rename event for any department by using the *OldText* parameter to set the name of the department back to the original name. A message box is also displayed informing the user that the name of the department cannot be changed.

```
Private Sub AnyDepartment_Rename(ByVal OldName As String)
    MsgBox "The name of the department cannot be changed."
    AnyDepartment.Name = OldName
End Sub
```

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

Size Property

Syntax *Department*.**Size**

Data Type Long (read/write)

Description The Size property controls the size of a department lane in a diagram. The property controls either the height or width, depending on how departments are oriented on the diagram (either horizontal or vertical, see the Departments.Orientation property). Values for the property are specified in twips (1440 twips = 1 inch).

Overall department sizing is controlled by several different properties (of different objects). Refer to the discussion of the Department object for information about the inter-relationships among the properties of various objects that can affect the look and size of a department.

Example The following example gets the ActiveDiagram object and then uses it to create a new department. The new Department object is used to change the size of the department to 3 inches.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Create a new department
Set igxDepartment = igxDiagram.Departments.AddDepartment _
    ("Department 1")
MsgBox "View the state of the diagram"
' Set the size of the department to 3 inches
igxDepartment.Size = 1440 * 3
MsgBox "View the state of the diagram"
```

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

TextRange Property

Syntax *Department*.**TextRange**([*First* As Long = 1], [*Last* As Long])

Data Type TextRange object (read-only, See Object Properties)

Description The TextRange property returns a TextRange object for the specified Department object. This property applies to the department name area as a whole, and marks either all or part of the text as being selected. The purpose of the TextRange object, is to provide control over selected text within the name area of the Department object.

The TextRange object lets you work with a range of text. The *First* and *Last* arguments specify the start and end positions of the text range. For example, specifying Paragraph1.TextRange(1,5) returns a TextRange that contains the first five characters of the paragraph. Specifying the property without providing the *First* and *Last* arguments returns a TextRange with all the characters in the paragraph. The *First* argument defaults to a value of 1, so to select from the first character of the paragraph only requires specifying the last character.

In addition, each Paragraph object contained within a Department object's name area has its own TextRange object that can be used to select either all or part of the paragraph.

Example The following example creates a department called "Department 1" in the active diagram. The Department's TextRange object is then used to select the word "Department". The the TextRange object's Font property is used to change the word to bold.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxTextRange As TextRange
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Create a new department.
Set igxDepartment = igxDiagram.Departments.AddDepartment _
    ("Department 1")
MsgBox "View the diagram"
' Get the TextRange object for Department 1 and select
' the word Department
Set igxTextRange = igxDepartment.TextRange(1, 10)
igxTextRange.Font.Bold = True
MsgBox "View the diagram"
```

See Also TextRange object

iGrafx API Object Hierarchy

{button Department object,JI(`igrafxrf.HLP',`Department_Object')}

Departments Object

The Departments object is a collection of individual Department objects. The Departments collection object is associated only with a Diagram object.

The Departments object provides the following functionality for working with Department objects.

- The ability to access any Department object that has been created in a diagram.
- The ability to determine how many Department objects are currently in the collection.
- The ability to set a number of formatting properties that affect all departments in a diagram, or that affect the printing of diagrams that contain departments.
- The ability to add a new Department object to a diagram.

Properties, Methods, and Events

All of the properties, methods, and events for the Departments object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	AddDepartment	
Count	Item	
DepartmentDividerLineFormat		
DepartmentFrameLineFormat		
EndMargin		
FillFormat		
HeaderSize		
LaneDividerLineFormat		
LaneMargin		
MinimumDeptSize		
NameAreaPosition		
Parent		
Orientation		
ShowHeadersEveryPage		

Related Topics

- Department object
- DepartmentNames object
- DepartmentRange object
- iGrafx API Object Hierarchy

AddDepartment Method

Syntax *Departments*.**AddDepartment** (*Name* As String, [*InsertPosition* As Integer]) As Department

Description The AddDepartment method adds a department to the Departments collection. The method returns a Department object as its result, and must be assigned to a variable of type Department.

The *Name* argument specifies the name of the department (for instance, Dept. 1 or Billing), and is displayed in the name area. The *InsertPosition* argument is optional, and is an integer value that specifies the location to place the department in relation to other departments already in the diagram. If there are no other departments in the diagram, then even if the *InsertPosition* argument is supplied, it is ignored. If you omit the *InsertPosition* argument, the new department is placed at the end of the list.

Example The following example gets the ActiveDiagram object and uses the AddDepartment method to add a department to the active diagram. The department is added in the second position.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
' Set the igxDiagram variable to the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add five new departments
Set igxDepartment = igxDepartments.AddDepartment("Shipping")
Set igxDepartment = igxDepartments.AddDepartment("Manufacturing")
Set igxDepartment = igxDepartments.AddDepartment("Research")
Set igxDepartment = igxDepartments.AddDepartment("Marketing")
Set igxDepartment = igxDepartments.AddDepartment("Sales")
MsgBox "View the state of the diagram"
' Add a new department at the second position.
Set igxDepartment = igxDepartments.AddDepartment _
   ("Second Department", 2)
MsgBox "View the state of the diagram"
```

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

DepartmentDividerLineFormat Property

Syntax *Departments*.**DepartmentDividerLineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The DepartmentDividerLineFormat property returns a LineFormat object that controls the formatting of the divider line that separates the name area from the process area of the department lanes. The divider line for every department is controlled by this property; you cannot format the divider lines on an individual department basis.

Example The following example creates four departments in the active diagram. It then sets the DepartmentDividerLineFormat property of the Departments collection to be a dashed line of size 2.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxDeptDivLineFmt As LineFormat
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add four new departments
Set igxDepartment = igxDepartments.AddDepartment("Shipping")
Set igxDepartment = igxDepartments.AddDepartment("Manufacturing")
Set igxDepartment = igxDepartments.AddDepartment("Research")
Set igxDepartment = igxDepartments.AddDepartment("Sales")
MsgBox "View the state of the diagram"
' Get the DepartmentDividerLineFormat
Set igxDeptDivLineFmt = igxDepartments.DepartmentDividerLineFormat
' Set the line characteristics
igxDeptDivLineFmt.Style = ixLineDashed
igxDeptDivLineFmt.Width = 40
MsgBox "View the state of the diagram"
```

See Also LineFormat object

iGrafx API Object Hierarchy

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

DepartmentFrameLineFormat Property

Syntax *Departments*.**DepartmentFrameLineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The DepartmentFrameLineFormat property returns a LineFormat object that controls the formatting of outer frame lines; that is, the outside border around all the department lanes. For example, if a diagram has four departments, the outer frame would be the left and right lines for all four departments, the top line of the first department, and the bottom line of the last department.

Example The following example uses the ActiveDiagram object to create a department on the active diagram. It then uses the Deparments collection from the ActiveDiagram to set the DepartmentFrameLineFormat.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxDeptFrameLineFmt As LineFormat
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add four new departments
Set igxDepartment = igxDepartments.AddDepartment("Shipping")
Set igxDepartment = igxDepartments.AddDepartment("Manufacturing")
Set igxDepartment = igxDepartments.AddDepartment("Research")
Set igxDepartment = igxDepartments.AddDepartment("Sales")
MsgBox "View the state of the diagram"
' Get the DepartmentFrameLineFormat
Set igxDeptFrameLineFmt = igxDepartments.DepartmentFrameLineFormat
' Set the line characteristics
igxDeptFrameLineFmt.Style = ixLineDashed
igxDeptFrameLineFmt.Width = 40
igxDeptFrameLineFmt.Color = vbRed
MsgBox "View the state of the diagram"
```

See Also LineFormat object

iGrafx API Object Hierarchy

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

EndMargin Property

Syntax *Departments*.**EndMargin**

Data Type Long (read/write)

Description The EndMargin property specifies the amount of white space (a margin) to leave between the last object in the process area (rightmost or bottom-most depending on the department orientation) and the department frame border. The property has an effect only if a shape is near the border of the first page, or if the department lane is longer than one page.

This property makes it possible to control how close a shape can be to right or bottom edge of the process area. Values for the property are specified in twips (1440 twips = 1 inch).

Example The following example uses the ActiveDiagram object to get the Departments collection object. The Departments collection object is then used to create a department and set the EndMargin property to 1/2 inch and the LaneMargin to 1/4 inch. A shape is created in Department 1. The shape is then moved to the right, and then down to show the EndMargin and LaneMargin settings.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxShape As Shape
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add the new department
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
' Set the EndMargin property to 1/2 inch
igxDepartments.EndMargin = 1440 / 2
' Set the LaneMargin property to 1/4 inch
igxDepartments.LaneMargin = 1440 / 4
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
'Move the shape to the right 5 inches
igxShape.DiagramObject.Left = 1440 * 6
MsgBox "View the state of the diagram"
'Move the shape down 1 inch
igxShape.DiagramObject.Top = 1440 * 2
MsgBox "View the state of the diagram"
```

See Also LaneMargin property

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

FillFormat Property

Syntax *Departments*.**FillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The FillFormat property returns a FillFormat object that controls the fill formatting of both the name area and the process area for all departments in the collection. The fill format controls whether the specified department uses a fill, and if so, what type of fill (solid, pattern, or gradient).

Fill formats specified at the Department object level override this property (see Department.FillFormat and Department.ProcessFillFormat). You can use this property to set a general fill format for all departments, then override it with specific formats for individual departments.

Example The following example creates three departments in the active diagram, sets the department margin properties and fill formatting for all three departments. It adds a shape into Department 1, and then changes the fill formatting in both the name area and the process area of Department 1.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxShape As Shape
Dim igxFillFormat As FillFormat
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
Set igxDepartment = igxDepartments.AddDepartment("Department 2")
Set igxDepartment = igxDepartments.AddDepartment("Department 3")
' Set EndMargin to 1/2 inch and LaneMargin to 1/4 inch
igxDepartments.EndMargin = 1440 / 2
igxDepartments.LaneMargin = 1440 / 4
MsgBox "View the state of the diagram"
' Get the FillFormat object at the Departments level
Set igxFillFormat = igxDepartments.FillFormat
' Set the fill properties
igxFillFormat.FillType = ixFillSolid
igxFillFormat.FillColor = vbBlue
MsgBox "View the state of the diagram"
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
Set igxDepartment = igxDepartments.Item(1)
igxDepartment.ProcessFillFormat.FillType = ixFillPattern
igxDepartment.ProcessFillFormat.PatternIndex = 10
MsgBox "View the state of the diagram"
igxDepartment.FillFormat.FillType = ixFillGradient
igxDepartment.FillFormat.GradientFormat.Type = ixGradientRadial
```

```
igxDepartment.FillFormat.FillColor = vbRed
igxDepartment.FillFormat.BackColor = vbYellow
MsgBox "View the state of the diagram"
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

HeaderSize Property

Syntax *Departments*.**HeaderSize**

Data Type Long (read/write)

Description The HeaderSize property specifies the width of the name area for all departments. You cannot adjust the header size for individual departments. Values for the property are specified in twips (1440 twips = 1 inch).

Example The following example creates three departments in the active diagram, and then sets the header size (name area) to 2 inches wide (or 2 inches long if the orientation is vertical).

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
Set igxDepartment = igxDepartments.AddDepartment("Department 2")
Set igxDepartment = igxDepartments.AddDepartment("Department 3")
MsgBox "View the state of the diagram"
' Set the header size to 2 inches
igxDepartments.HeaderSize = 1440 * 2
MsgBox "View the state of the diagram"
```

Item Method

Syntax *Departments*.**Item**(*Index* As Integer) As Department

Description The Item method returns the Department object at the specified *Index* from the Departments collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Department. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example uses the Departments.Item method to access each of the three departments that have been added to the active diagram. As each department is accessed, its fill properties are altered.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxShape As Shape
Dim igxFillFormat As FillFormat
Dim iCount As Integer
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
Set igxDepartment = igxDepartments.AddDepartment("Department 2")
Set igxDepartment = igxDepartments.AddDepartment("Department 3")
MsgBox "View the state of the diagram"
' Get the FillFormat object at the Departments level
Set igxFillFormat = igxDepartments.FillFormat
' Set the fill properties
igxFillFormat.FillType = ixFillSolid
igxFillFormat.FillColor = vbBlue
MsgBox "View the state of the diagram"
For iCount = 1 To igxDepartments.Count
    Select Case igxDepartments.Item(iCount).DepartmentIndex
        Case 1:
            Set igxDepartment = igxDepartments.Item(iCount)
            igxDepartment.ProcessFillFormat.FillType = ixFillPattern
            igxDepartment.ProcessFillFormat.PatternIndex = 10
            igxDepartment.ProcessFillFormat.BackColor = vbRed
            MsgBox "View the state of the diagram"
            igxDepartment.FillFormat.FillType = ixFillGradient
            igxDepartment.FillFormat.GradientFormat.Type = _
                ixGradientRadial
            igxDepartment.FillFormat.FillColor = vbRed
            igxDepartment.FillFormat.BackColor = vbYellow
            MsgBox "View the state of the diagram"
        Case 2:
            Set igxDepartment = igxDepartments.Item(iCount)
            igxDepartment.ProcessFillFormat.FillType = ixFillSolid
            igxDepartment.ProcessFillFormat.FillColor = vbYellow
            MsgBox "View the state of the diagram"
```

```
        Case 3:
            igxDepartment.ProcessFillFormat.FillType = ixFillPattern
            igxDepartment.ProcessFillFormat.PatternIndex = 3
            igxDepartment.ProcessFillFormat.BackColor = vbGreen
            MsgBox "View the state of the diagram"
            igxDepartment.FillFormat.FillType = ixFillGradient
            igxDepartment.FillFormat.GradientFormat.Type = _
                ixGradientSquare
            igxDepartment.FillFormat.FillColor = vbBlue
            igxDepartment.FillFormat.BackColor = vbWhite
            MsgBox "View the state of the diagram"
    End Select
Next iCount
```

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

LaneDividerLineFormat Property

Syntax *Departments*.**LaneDividerLineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The LaneDividerLineFormat property returns a LineFormat object that controls the formatting of the lane divider lines that separate the department lanes. All lane divider lines are controlled by this property; you cannot format the lane divider lines on an individual department basis.

There is no lane divider line if there is only one department in the diagram. Furthermore, the top line of the first department and the bottom line of the last department are not lane divider lines (they are department frame border lines—see the DepartmentFrameLineFormat property).

Example The following example uses the ActiveDiagram object to create three departments in the active diagram. It then sets the LaneDividerLineFormat to be a dashed yellow line with the width set to 3. As a variation, remove the creation of Departments 2 and 3 and run the example again. Note that the formatting of the lane divider has no effect because there is only one department.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxLaneDivLineFmt As LineFormat
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
Set igxDepartment = igxDepartments.AddDepartment("Department 2")
Set igxDepartment = igxDepartments.AddDepartment("Department 3")
MsgBox "View the state of the diagram"
' Get the LaneDividerLineFormat
Set igxLaneDivLineFmt = igxDepartments.LaneDividerLineFormat
' Set properties for the lane divider lines
igxLaneDivLineFmt.Style = ixLineDashed
igxLaneDivLineFmt.Width = 60
igxLaneDivLineFmt.Color = vbYellow
MsgBox "View the state of the diagram"
```

See Also DepartmentFrameLineFormat property

LineFormat object

iGrafx API Object Hierarchy

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

LaneMargin Property

Syntax *Departments*.**LaneMargin**

Data Type Long (read/write)

Description The LaneMargin property specifies the amount of white space (a margin) to leave between the edge of a shape and

- The bottom edge of the department lane if the orientation is horizontal.
- The left edge of the department lane if the orientation is vertical.

This property makes it possible to control how close a shape can be to the bottom or left edge of the process area. Values for the property are specified in twips (1440 twips = 1 inch).

Example The following example uses the ActiveDiagram object to get the Departments collection object. The Departments collection object is then used to create a department and set the EndMargin property to 1/2 inch and the LaneMargin to 1/4 inch. A shape is created in Department 1. The shape is then moved to the right, and then down to show the EndMargin and LaneMargin settings.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxShape As Shape
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add the new department
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
' Set the EndMargin property to 1/2 inch
igxDepartments.EndMargin = 1440 / 2
' Set the LaneMargin property to 1/4 inch
igxDepartments.LaneMargin = 1440 / 4
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
' Move the shape to the right 5 inches
igxShape.DiagramObject.Left = 1440 * 6
MsgBox "View the state of the diagram"
' Move the shape down 1 inch
igxShape.DiagramObject.Top = 1440 * 2
MsgBox "View the state of the diagram"
```

See Also EndMargin property

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

MinimumDeptSize Property

Syntax *Departments*.**MinimumDeptSize**

Data Type Long (read/write)

Description The MinimumDeptSize specifies the minimum size for all department lanes (the height for horizontal orientation and the width for vertical orientation). Attempts to set the size smaller than this minimum, either programmatically with the Department.Size property or manually, are ignored. Values for the property are specified in twips (1440 twips = 1 inch).

Example The following example creates three departments in the active diagram. It then sets the MinimumDeptSize property to 2 inches, and then prints the current size of each department to the Immediate window. A shape is then created in Department 1. The code then attempts to set the size of Department to 1 inch, which is less than the minimum. Finally, the current size of each department is again printed to the Immediate window.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxShape As Shape
Dim iCount As Integer
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
Set igxDepartment = igxDepartments.AddDepartment("Department 2")
Set igxDepartment = igxDepartments.AddDepartment("Department 3")
MsgBox "View the state of the diagram"
igxDepartments.MinimumDeptSize = 1440 * 2
MsgBox "View the state of the diagram"
' Print to the Immediate window the size of all departments
For iCount = 1 To igxDepartments.Count
   Debug.Print igxDepartments.Item(iCount).DepartmentName _
       & " Size is " & igxDepartments.Item(iCount).Size
Next iCount
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
Set igxDepartment = igxDepartments.Item(1)
igxDepartment.Size = 1440
MsgBox "View the state of the diagram"
For iCount = 1 To igxDepartments.Count
   Debug.Print igxDepartments.Item(iCount).DepartmentName _
       & " Size is " & igxDepartments.Item(iCount).Size
Next iCount
```

NameAreaPosition Property

Syntax *Departments*.**NameAreaPosition**

Data Type IxDeptNamePosition enumerated constant (read/write)

Description The NameAreaPosition property controls the visibility and positioning of all the department name areas on a diagram. The value of this property affects all departments in a diagram; the visibility and position of the department name areas cannot be controlled individually. When you create a department on a diagram with the AddDepartment method, this property defaults to ixDeptNameOnLeft.

The IxDeptNamePosition constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant	Description
0	ixDeptNameHidden	Departments are hidden, and do not appear on the diagram.
1	ixDeptNameOnLeft	All department names appear on the left side of the diagram.
2	ixDeptNameOnRight	All department names appear on the right side of the diagram.
3	ixDeptNameBoth	All department names appear on both sides of the diagram.

Note that the constant names are biased toward the horizontal department orientation. If the orientation is vertical, then the following are true: Left is equivalent to Top, Right is equivalent to Bottom, Both is equivalent to Top and Bottom.

Example The following example creates three departments in the active diagram, and places a shape in the first department. It then cycles through each possible setting for the name area position (the default is Left).

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxShape As Shape
Dim iCount As Integer
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
Set igxDepartment = igxDepartments.AddDepartment("Department 2")
Set igxDepartment = igxDepartments.AddDepartment("Department 3")
MsgBox "View the state of the diagram"
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
' Cycle through all the settings for the name area position
```

```
For iCount = 0 To 3
   igxDepartments.NameAreaPosition = iCount
   MsgBox "View the state of the diagram"
Next iCount
```

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

Orientation Property

Syntax *Departments*.**Orientation**

Data Type IxDeptOrient enumerated constant (read/write)

Description The Orientation property controls how all department lanes are oriented in a diagram. Department lane orientation cannot be controlled for individual departments.

The IxDeptOrient constant defines the valid values for this property, which are listed in the following table (ixDeptOrientHorz is the default).

Value	Name of Constant
0	ixDeptOrientHorz
1	ixDeptOrientVert

This property should be set prior to adding departments to a diagram; you cannot change the orientation of existing departments later. Furthermore, the orientation of the first department you add to a diagram is the one you must use for all subsequent departments you add. Attempting to add a department with the other orientation does not work, and the Departments.AddDepartment statement is ignored.

The orientation of department lanes can affect other properties that relate to departments (refer to the See Also topics).

Example The following example sets the department orientation to vertical and creates three departments in the active diagram, and places a shape in the first department. It then cycles through each possible setting for the name area position (the default is Left, or Top when the orientation is vertical).

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxShape As Shape
Dim iCount As Integer
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Set the orientation of departments to Vertical
' This must be done prior to creating the departments
igxDepartments.Orientation = ixDeptOrientVert
' Add three new departments
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
Set igxDepartment = igxDepartments.AddDepartment("Department 2")
Set igxDepartment = igxDepartments.AddDepartment("Department 3")
MsgBox "View the state of the diagram"
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
' Cycle through all the settings for the name area position
```

```
For iCount = 0 To 3
   igxDepartments.NameAreaPosition = iCount
   MsgBox "View the state of the diagram"
Next iCount
```

See Also

Department object

DiagramObject object

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

ShowHeadersEveryPage Property

Syntax *Departments*.**ShowHeadersEveryPage**[= {**True | False**}]

Data Type Boolean (read/write)

Description The ShowHeadersEveryPage property specifies whether to print the department name area of all departments in a diagram on every page. The property only affects diagram printing, not the diagram display through the user interface. The effect of setting this property can be seen by using the File—Print Preview command.

Note The ShowHeadersEveryPage property is dependent on the setting for titles. If the ShowHeadersEveryPage property is set to True, it only has an effect if the PageLayout.PageTitleMode property is set to ixPerPage. The title mode also can be controlled through the user interface by going to the **File->Page Setup->Options** menu item, and checking the Titles->Per Page option.

Example The following example creates a diagram with several departments, and shapes on two pages. It then sets the ShowHeadersEveryPage property to True, and show Print Preview to display the result.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
Dim igxShape As Shape
Dim iCount As Integer
' Get the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
igxDepartments.Orientation = ixDeptOrientHorz
igxDepartments.HeaderSize = 2160
igxDepartments.DepartmentDividerLineFormat.Style = ixLineNormal
igxDepartments.DepartmentDividerLineFormat.Color = vbRed
igxDepartments.LaneDividerLineFormat.Style = ixLineNormal
igxDepartments.LaneDividerLineFormat.Color = vbYellow
' Add three new departments
Set igxDepartment = igxDepartments.AddDepartment("Department 1")
Set igxDepartment = igxDepartments.AddDepartment("Department 2")
Set igxDepartment = igxDepartments.AddDepartment("Department 3")
MsgBox "View the state of the diagram"
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440 * 8, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440 * 2)
```

```
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440 * 8, 1440 * 2)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
igxDepartments.ShowHeadersEveryPage = True
Application.ExecuteCommand (ixFilePrintPreview)
```

See Also PageLayout.PageTitleMode property

{button Departments object,JI(`igrafxrf.HLP',`Departments_Object')}

DepartmentNames Object

The DepartmentNames object is an array of strings (a collection) that stores department names. That is, when a Department is created, whatever name is specified for the DepartmentName property is automatically stored by iGrafx Professional in the DepartmentNames object. The ordering of names in the array is based on creation order.

A DepartmentNames collection is associated only with a Document object. The DepartmentNames object allows the developer to:

- Access all of the department names in a document (across multiple diagrams, potentially)
- Determine how many department names exist in the document

If multiple diagrams use the same department name, then the name is only listed once in the DepartmentNames collection object.

The following example gets the ActiveDocument object and then uses it to retrieve the DepartmentNames collection object.

```
' Dimension the variables
Dim igxDocument As Document
Dim igxDepartmentNames As DepartmentNames
' Get the ActiveDiagram object
Set igxDocument = Application.ActiveDocument
' Get the DepartmentNames object
Set igxDepartmentNames = igxDocument.DepartmentNames
```

Properties, Methods, and Events

All of the properties, methods, and events for the DepartmentNames object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
Parent		

Item Method

Syntax *DepartmentNames*.**Item**(*Index* As Integer) As String

Description The Item method returns the department name string located at the specified *Index* from the DepartmentNames collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type String. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example adds five departments to the active diagram. It then creates a new diagram and adds two departments that have the same names as two of the departments in Diagram 1. It then uses the DepartmentNames collection to search for specific department names in Diagram 1. When found, each department has some formatting characteristics set. Finally, the entire contents of the DepartmentNames colelction is printed to the Output window.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDepartment As Department
Dim igxDepartments As Departments
' Set the igxDiagram variable to the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add five new departments
Set igxDepartment = igxDepartments.AddDepartment("Shipping")
Set igxDepartment = igxDepartments.AddDepartment("Manufacturing")
Set igxDepartment = igxDepartments.AddDepartment("Research")
Set igxDepartment = igxDepartments.AddDepartment("Marketing")
Set igxDepartment = igxDepartments.AddDepartment("Sales")
MsgBox "View the state of the diagram"
' Create a new diagram and add two departments to it
Set igxDiagram = Application.ActiveDocument.Diagrams.Add("Diagram2")
igxDiagram.ActivateDiagram
igxDiagram.Departments.AddDepartment ("Sales")
igxDiagram.Departments.AddDepartment ("Shipping")
MsgBox "View the state of the diagram"
' Switch back to the first diagram
Set igxDiagram = ActiveDocument.Diagrams.Item(1)
igxDiagram.ActivateDiagram
MsgBox "View the state of the diagram"
' Add fill formatting to the departments in the first document
' based on finding the department name
For iCount = 1 To igxDepartments.Count
   Select Case ActiveDocument.DepartmentNames.Item(iCount)
       Case "Shipping":
           Set igxDepartment = igxDepartments.Item(iCount)
           igxDepartment.ProcessFillFormat.FillType = ixFillPattern
           igxDepartment.ProcessFillFormat.PatternIndex = 10
           MsgBox "View the state of the diagram"
       Case "Manufacturing":
           Set igxDepartment = igxDepartments.Item(iCount)
           igxDepartment.ProcessFillFormat.FillType = ixFillSolid
           igxDepartment.ProcessFillFormat.FillColor = vbRed
           MsgBox "View the state of the diagram"
```

```
        Case "Research":
            Set igxDepartment = igxDepartments.Item(iCount)
            igxDepartment.ProcessFillFormat.FillType = ixFillGradient
            igxDepartment.ProcessFillFormat.FillColor = vbBlue
            igxDepartment.ProcessFillFormat.BackColor = vbWhite
            igxDepartment.ProcessFillFormat.GradientFormat.Type = _
                ixGradientRadial
            igxDepartment.ProcessFillFormat.GradientFormat.XOrigin = 25
            MsgBox "View the state of the diagram"
        Case "Marketing":
            Set igxDepartment = igxDepartments.Item(iCount)
            igxDepartment.ProcessFillFormat.FillType = ixFillSolid
            igxDepartment.ProcessFillFormat.FillColor = vbGreen
            MsgBox "View the state of the diagram"
        Case "Sales"
            Set igxDepartment = igxDepartments.Item(iCount)
            igxDepartment.FillFormat.FillType = ixFillPattern
            igxDepartment.FillFormat.PatternIndex = 20
            MsgBox "View the state of the diagram"
    End Select
Next iCount
' Print the DepartmentNames list in the Output window
For iCount = 1 To ActiveDocument.DepartmentNames.Count
    Output ActiveDocument.DepartmentNames.Item(iCount)
Next iCount
```

{button DepartmentNames object,JI(`igrafxrf.HLP',`DepartmentNames_Object')}

DepartmentRange Object

The DepartmentRange object is a collection associated only with the Shape object, and contains the list of departments to which a shape belongs. The collection stores Department objects. Its purpose is to store all the Department objects to which a Shape object is associated.

The DepartmentRange object provides the following functionality:

- The ability to access any Department object that is associated with a shape.
- The ability to determine how many Department objects are associated with a shape (currently in the collection).
- The ability to add a new Department association with a shape.
- The ability to remove an existing Department association with a shape.

If a shape is drawn inside the boundaries of an existing department, that department is automatically added to the DepartmentRange collection. You can verify this by checking the DepartmentRange collection.

Both the DepartmentRange and ExcludedDepartmentNames objects are object properties of a shape, and they are inter-related. For example, if a shape is drawn so it belongs to three departments, the DepartmentRange object contains those three Department objects in the collection. If one of those department names is added to the ExcludedDepartmentNames object for the shape, then that Department object is removed from the DepartmentRange collection.

Adding departments to the collection with the Add method (and removing them with the Remove method) also has inter-relationships with the ExcludedDepartmentNames object. For more information, refer to the descriptions of those methods.

Individual Department objects are accessed with the Item method; therefore, the developer can apply changes to all the departments or just to some subset by looping through the collection and checking for certain property values, etc.

The following example uses the ActiveDiagram object to create a shape on the active diagram. The Shape object is then used to get the DepartmentRange object.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxDepartmentRange As DepartmentRange
' Set the igxDiagram variable to the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create a shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the shape's DepartmentRange object
Set igxDepartmentRange = igxShape.DepartmentRange
```

Properties, Methods, and Events

All of the properties, methods, and events for the DepartmentRange object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent	Remove	

Add Method

Syntax *DepartmentRange*.**Add (***DepartmentToAdd* As Department) As Boolean

Description The Add method adds the department specified by the *DepartmentToAdd* argument to the DepartmentRange collection. The argument value must be a variable of type Department. The method returns a Boolean result, indicating success or failure, and must be assigned to a variable of type Boolean. A value of True indicates that the Add operation was successful.

This method adds only the specified department, and affects the diagram by extending the shape into the process area of the added department. If the added department is not adjacent to the last department the shape is associated with, then any departments in between are added to the ExcludedDepartmentNames object of the shape (see the example).

If a shape is drawn inside the boundaries of an existing department, you do not need to explicitly add the department to the DepartmentRange collection; that is done automatically when the shape is created.

Example The following example creates three departments in the active diagram. It then adds a shape so it is located within the boundaries of the first department. This associates the shape with the department, and that department is added to the DepartmentRange collection. Then the shape is associated with the other two departments by adding them to the DepartmentRange collection. Then the second department is removed. Since Department 2 separates the other two departments, the only way to represent the shape as belonging to Departments 1 and 3 is to exclude Department 2; therefore, Department 2 is added to the ExcludedDepartmentNames collection. Next, Department 1 is removed from the DepartNames collection, and it, too, is added to the ExcludedDepartmentNames collection. Finally, the shape is moved so it is no longer within the area of Department 1: this action removes the shape's association with Department 1.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxDepartmentRange As DepartmentRange
Dim igxExclDepartmentNames As ExcludedDepartmentNames
Dim igxDepartment1 As Department
Dim igxDepartment2 As Department
Dim igxDepartment3 As Department
Dim igxDepartments As Departments
Dim lShapeBottom As Long
' Set the igxDiagram variable to the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment1 = igxDepartments.AddDepartment("Department 1")
Set igxDepartment2 = igxDepartments.AddDepartment("Department 2")
Set igxDepartment3 = igxDepartments.AddDepartment("Department 3")
' Create a shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the diagram"
' Get the new shape's DepartmentRange object
Set igxDepartmentRange = igxShape.DepartmentRange
MsgBox "DepartmentRange object for the shape contains " _
    & igxDepartmentRange.Count & " departments. " _
```

```
    & "That department is " & igxDepartmentRange.Item(1).DepartmentName
' The first department is already a member of the
' DepartmentRange collection, so add the third
' department to the department range
fSuccess = igxDepartmentRange.Add(igxDepartment2)
If (fSuccess) Then
    MsgBox "Associated the shape with " _
    & igxDepartmentRange.Item(2).DepartmentName
End If
' Add the second department to the department range
fSuccess = igxDepartmentRange.Add(igxDepartment3)
If (fSuccess) Then
    MsgBox "Associated the shape with " _
        & igxDepartmentRange.Item(3).DepartmentName
End If
MsgBox "The shape is now associated with " _
    & igxDepartmentRange.Count & " departments"
' Display the message box to remove the second department
MsgBox "Click OK to remove the second department."
' Remove the second department from the department range
fSuccess = igxDepartmentRange.Remove(igxDepartment2)
MsgBox "Removed Dept 2 from the shape's department range"
' Get the ExcludedDepartmentRange object
Set igxExclDepartmentNames = igxShape.ExcludedDepartmentNames
' Remove the second department from the excluded list
fSuccess = igxExclDepartmentNames.Remove _
    (igxDepartment2.DepartmentName)
MsgBox "Removed Dept 2 from the shape's excluded departments list"
' Display the message box to remove the first department
MsgBox " Click OK to remove the first department."
' Remove the first department from the department range
fSuccess = igxDepartmentRange.Remove(igxDepartment1)
MsgBox "First department removed from the department range"
' Remove the first department from the excluded list
fSuccess = igxExclDepartmentNames.Remove _
    (igxDepartment1.DepartmentName)
MsgBox "First department removed from the excluded departments list"
' Remove the association with department 1 by moving the shape
' out of the area of the department
igxShape.DiagramObject.Top = 3000
' Reset the shape's height
igxShape.DiagramObject.Height = 1440 * 2.5
MsgBox "The shape is now associated with " _
    & igxDepartmentRange.Count & " departments"
```

See Also Remove method

{button DepartmentRange object,JI(`igrafxrf.HLP',`DepartmentRange_Object')}

Item Method

Syntax *DepartmentRange*.**Item**(*Index* As Integer) As Department

Description The Item method returns the Department object at the specified *Index* from the DepartmentRange collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Department. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example Refer to the Example sections of the Add and Remove methods.

See Also Add method

Remove method

{button DepartmentRange object,JI(`igrafxrf.HLP',`DepartmentRange_Object')}

Remove Method

Syntax *DepartmentRange*.**Remove (***DepartmentToRemove* As Department) As Boolean

Description The Remove method removes the department specified by the *DepartmentToRemove* argument from the DepartmentRange collection. The argument value must be a variable of type Department. The method returns a Boolean result, indicating success or failure, and must be assigned to a variable of type Boolean. A value of True indicates that the remove operation was successful.

This method removes only the specified department. The result to the shape depends on which department is removed. If the removed department is the top or bottom department the shape is associated with, then the shape is resized and drawn only in the lanes of the remaining departments in the DepartmentRange collection. If the removed department is in the interior (for instance, Department 2 in the example), then the removed department is automatically added to the ExcludedDepartmentNames object, and is drawn accordingly on the diagram.

Example The following example creates three departments in the active diagram. It then adds a shape so it is located within the boundaries of the first department. This associates the shape with the department, and that department is added to the DepartmentRange collection. Then the shape is associated with the other two departments by adding them to the DepartmentRange collection. Then the second department is removed. Since Department 2 separates the other two departments, the only way to represent the shape as belonging to Departments 1 and 3 is to exclude Department 2; therefore, Department 2 is added to the ExcludedDepartmentNames collection.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxDepartmentRange As DepartmentRange
Dim igxExclDepartmentNames As ExclDepartmentNames
Dim igxDepartment1 As Department
Dim igxDepartment2 As Department
Dim igxDepartment3 As Department
Dim igxDepartments As Departments
' Set the igxDiagram variable to the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment1 = igxDepartments.AddDepartment("Department 1")
Set igxDepartment2 = igxDepartments.AddDepartment("Department 2")
Set igxDepartment3 = igxDepartments.AddDepartment("Department 3")
' Create a shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
' Set the fill type of the shape
igxShape.FillType = ixFillSolid
' Set the fill color of the shape
igxShape.FillColor = vbGreen
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
' Get the new shape's DepartmentRange object
Set igxDepartmentRange = igxShape.DepartmentRange
MsgBox "DepartmentRange object for the shape contains " _
    & igxDepartmentRange.Count & " departments. " _
```

```
    & "That department is " & igxDepartmentRange.Item(1).DepartmentName
' The first department is already a member of the
' DepartmentRange collection, so add the second
' department to the department range
fSuccess = igxDepartmentRange.Add(igxDepartment2)
If (fSuccess) Then
    MsgBox "Associated the shape with " _
        & igxDepartmentRange.Item(2).DepartmentName
End If
' Add the third department to the department range
fSuccess = igxDepartmentRange.Add(igxDepartment3)
If (fSuccess) Then
    MsgBox "Associated the shape with " _
        & igxDepartmentRange.Item(3).DepartmentName
End If
MsgBox "The shape is now associated with " _
    & igxDepartmentRange.Count & " departments"
' Display the message box to remove the second department
MsgBox " Click OK to remove the second department."
' Remove the second department from the department range
fSuccess = igxDepartmentRange.Remove(igxDepartment2)
MsgBox "View the state of the diagram"
```

Variation

Add the following lines of code to the example. Also, try varying which department you remove from the DepartmentNames collection and observe what happens to the shape, and whether there are any items in the ExcludedDepartmentNames collection.

```
' Get the ExcludedDepartmentRange object.
Set igxExclDepartmentNames = igxShape.ExcludedDepartmentNames
fSuccess = igxExclDepartmentNames.Remove _
   (igxDepartment2.DepartmentName)
MsgBox "View the state of the diagram"
```

See Also

Add method

ExcludedDepartmentNames object

{button DepartmentRange object,JI(`igrafxrf.HLP',`DepartmentRange_Object')}

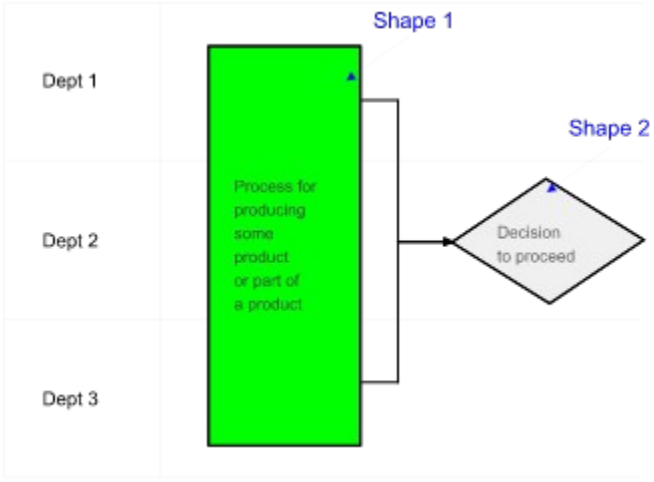
ExcludedDepartmentNames Object

The ExcludedDepartmentNames object provides a way to exclude a shape from being associated with one or more departments if that shape is drawn so that it spans more than one department. The object stores an array of strings, which are the names of departments; that is, the value of the DepartmentName property.

The ExcludedDepartmentNames collection is associated only with a Shape object. The object allows the developer to:

- Access all of the excluded department names for a shape
- Determine how many department names are excluded from a shape
- Add the name of a department to exclude from a shape
- Remove the exclusion of a department name from a shape

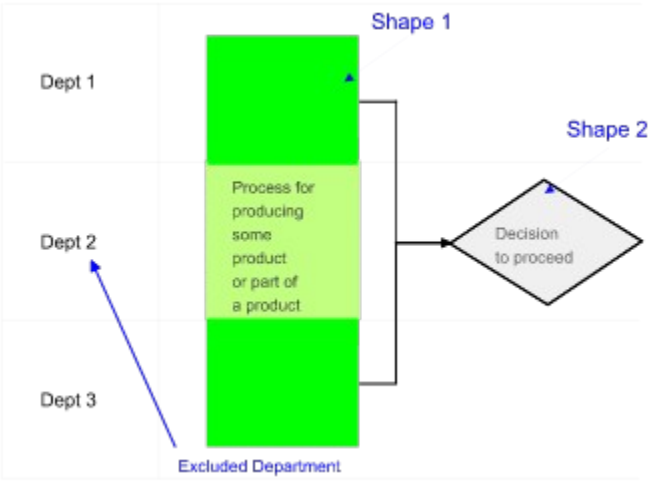
For example, in a process diagram you might have a case where a decision or procedure is either owned or performed by multiple departments in your organization. However, due to other process associations, these departments cannot be drawn next to each other. This concept is illustrated in the following diagram.



For this example, Dept 1 and Dept 3 own the production of some deliverable, and Dept 2 owns the decision whether to proceed with further development. By default in iGrafx Professional, Shape 1 would be associated with all three departments, but you only want it associated with departments 1 and 3. Excluding Dept 2 from Shape 1 is done with the Add method, as follows.

```
Shape1.ExcludedDepartmentNames.Add "Dept 2"
```

After excluding Dept 2 from Shape 1, the diagram should look as shown below.



The ExcludedDepartmentNames object has inter-relationships with the DepartmentRange object. Refer to the documentation of the DepartmentRange object for details.

Another use for this object might be when you want to perform some action on all shapes that belong to a specific department. Since, by default, Shape1 is associated with Dept 2 but is not exclusive to Dept 2, you may want to exclude it from being affected by the action you want to perform.

The following example creates a shape on the active diagram and gets the ExcludedDepartmentNames object from the Shape object.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxExcludedDepartmentNames As ExcludedDepartmentNames
' Set the igxDiagram variable to the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the ExcludedDepartmentNames object from the shape
Set igxExcludedDepartmentNames = igxShape.ExcludedDepartmentNames
```

Properties, Methods, and Events

All of the properties, methods, and events for the ExcludedDepartmentNames object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent	Remove	

Related Topics

Shape object

DepartmentRange object

iGrafx API Object Hierarchy

Add Method

Syntax *ExcludedDepartmentNames*.**Add**(*DepartmentName* As String) As Boolean

Description The Add method adds a department name to the ExcludedDepartmentNames collection for a specific shape. The method returns a Boolean result indicating success or failure, and must be assigned to a variable of type Boolean. A value of True indicates that the Add operation was successful.

The *DepartmentName* argument must be a literal string or a string variable, and a valid department name.

Example The following example creates three departments in the active diagram. It then adds a shape so it is located within the boundaries of the first department. This associates the shape with the department, and that department is added to the DepartmentRange collection. Then the shape is associated with the other two departments by adding them to the DepartmentRange collection. Then the second department is excluded by adding it to the ExcludedDepartmentNames collection.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxDepartmentRange As DepartmentRange
Dim igxExclDepartmentNames As ExcludedDepartmentNames
Dim igxDepartment1 As Department
Dim igxDepartment2 As Department
Dim igxDepartment3 As Department
Dim igxDepartments As Departments
Dim fSuccess As Boolean
' Set the igxDiagram variable to the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment1 = igxDepartments.AddDepartment("Department 1")
Set igxDepartment2 = igxDepartments.AddDepartment("Department 2")
Set igxDepartment3 = igxDepartments.AddDepartment("Department 3")
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the state of the diagram"
' Get the new shape's DepartmentRange object
Set igxDepartmentRange = igxShape.DepartmentRange
MsgBox "DepartmentRange object for the shape contains " _
   & igxDepartmentRange.Count & " departments. " _
   & "That department is " _
    & igxDepartmentRange.Item(1).DepartmentName
' The first department is already a member of the
' DepartmentRange collection, so add the second
' department to the department range
fSuccess = igxDepartmentRange.Add(igxDepartment2)
If (fSuccess) Then
   MsgBox "Associated the shape with " _
        & igxDepartmentRange.Item(2).DepartmentName
```

```
End If
' Add the third department to the department range
fSuccess = igxDepartmentRange.Add(igxDepartment3)
If (fSuccess) Then
   MsgBox "Associated the shape with " _
       & igxDepartmentRange.Item(3).DepartmentName
End If
MsgBox "The shape is now associated with " _
   & igxDepartmentRange.Count & " departments"
' Get the ExcludedDepartmentRange object.
Set igxExclDepartmentNames = igxShape.ExcludedDepartmentNames
' Exclude the second department
fSuccess = igxExclDepartmentNames.Add(igxDepartment2.DepartmentName)
If (fSuccess) Then
   MsgBox "The second department is now excluded."
Else
   MsgBox "Add operation of ExcludedDepartmentNames object failed."
End If
```

See Also

Remove method

DepartmentNames object

DepartmentRange object

{button ExcludedDepartmentNames object,JI(`igrafxrf.HLP',`ExcludedDepartmentNames_Object')}

Item Method

Syntax *ExcludedDepartmentNames*.**Item**(*Index* As Integer) As String

Description The Item method returns a string, the department name, that is located at the specified *Index* from the ExcludedDepartmentNames collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type String. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example Refer to the examples for the Add and Remove methods, which both show the use of the Item method.

See Also Add method

Remove method

{button ExcludedDepartmentNames object,JI(`igrafxrf.HLP',`ExcludedDepartmentNames_Object')}

Remove Method

Syntax *ExcludedDepartmentNames*.**Remove** (*DepartmentName* As String) As Boolean

Description The Remove method removes a department name from the ExcludedDepartmentNames collection for a specific shape. The method returns a Boolean result indicating the success or failure, and must be assigned to a variable of type Boolean. A value of True indicates that the remove operation was successful.

The *DepartmentName* argument must be a literal string or a string variable, and a valid department name. If the department name is not in the collection, the method fails (the Boolean result is False), and has no effect. It is advisable to write your own error routine to trap the result of False.

Example The following example creates three departments in the active diagram. It then adds a shape so it is located within the boundaries of the first department. This associates the shape with the department, and that department is added to the DepartmentRange collection. Then the shape is associated with the other two departments by adding them to the DepartmentRange collection. Then the second department is excluded by adding it to the ExcludedDepartmentNames collection. Finally, the second department is re-associated with the shape by removing it from the ExcludedDepartmentNames collection.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxDepartmentRange As DepartmentRange
Dim igxExclDepartmentNames As ExcludedDepartmentNames
Dim igxDepartment1 As Department
Dim igxDepartment2 As Department
Dim igxDepartment3 As Department
Dim igxDepartments As Departments
Dim fSuccess As Boolean
' Set the igxDiagram variable to the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Get the Departments collection object
Set igxDepartments = igxDiagram.Departments
' Add three new departments
Set igxDepartment1 = igxDepartments.AddDepartment("Department 1")
Set igxDepartment2 = igxDepartments.AddDepartment("Department 2")
Set igxDepartment3 = igxDepartments.AddDepartment("Department 3")
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
' Set shape fill properties
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbGreen
MsgBox "View the diagram"
' Get the new shape's DepartmentRange object
Set igxDepartmentRange = igxShape.DepartmentRange
MsgBox "DepartmentRange object for the shape contains " _
   & igxDepartmentRange.Count & " departments. " _
   & "That department is " _
    & igxDepartmentRange.Item(1).DepartmentName
' The first department is already a member of the
' DepartmentRange collection, so add the second
' department to the department range
fSuccess = igxDepartmentRange.Add(igxDepartment2)
```

```
If (fSuccess) Then
   MsgBox "Associated the shape with " _
       & igxDepartmentRange.Item(2).DepartmentName
End If
' Add the third department to the department range
fSuccess = igxDepartmentRange.Add(igxDepartment3)
If (fSuccess) Then
   MsgBox "Associated the shape with " _
       & igxDepartmentRange.Item(3).DepartmentName
End If
MsgBox "The shape is now associated with " _
   & igxDepartmentRange.Count & " departments"
' Get the ExcludedDepartmentRange object
Set igxExclDepartmentNames = igxShape.ExcludedDepartmentNames
' Exclude the second department
fSuccess = igxExclDepartmentNames.Add(igxDepartment2.DepartmentName)
' Display the message box
MsgBox "The second department is now excluded." & _
   " Click OK to not exclude the second department."
' Remove the second department from the excluded list
fSuccess = igxExclDepartmentNames.Remove _
   (igxDepartment2.DepartmentName)
MsgBox "View the diagram"
```

See Also

Add method

DepartmentRange object

{button ExcludedDepartmentNames object,JI(`igrafxrf.HLP',`ExcludedDepartmentNames_Object')}

BlockFormat Object

The BlockFormat object controls the formatting of the text block (or area) associated with the following objects:

- A Shape object, through the TextBlock or ChildTextBlock objects. The TextBlock object (there is only one per shape) and all ChildTextBlock objects (there can be zero or more per shape) have there own distinct BlockFormat objects for controlling text formatting.
- A TextGraphicObject object
- A Department object
- A HeaderFooter object

The BlockFormat object controls formatting of the text block, which is the container into which text is placed. With this object, you can specify fills, borders, orientation, etc., for the text block. The actual text is controlled by other properties, depending on the parent object (refer to the iGrafx Professional API Object Hierarchy).

The BlockFormat object can be assigned to another BlockFormat object as a whole. That is, the following assignment statement is valid:

```
Set Shape1.TextBlock.BlockFormat = Shape2.TextBlock.BlockFormat
```

The following example shows a typical method of accessing the BlockFormat object of a Shape object. The code creates a shape on the active document, and then through the TextBlock object, gets the BlockFormat object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBlkFmt As BlockFormat
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the BlockFormat object
Set igxBlkFmt = igxShape.TextBlock.BlockFormat
```

Important

The intent of the BlockFormat object is to provide control over formatting the text areas of shapes, graphics, etc., such as text blocks and child text blocks. Many of these same formatting capabilities are available from other objects, such as a shape, which can produce conflicts or unpredicatable behavior in situations where the size of the shape and the size of the text block are the same. The BlockFormat properties typically override the same properties at the Shape or TextGraphicObject level. Some of these situations are documented in the descriptions of the BlockFormat properties. To get predicatable behavior, it is best to code according to the following rule:

> *If the size of the shape, text graphic object, etc. is the same as the text block area (that is, you have not adjusted the size of the text block relative to the shape), then always use the Shape object's formatting properties.*

However, there are some interesting effects you can obtain by specifying line properties (typically of different sizes or styles) for both the text block and a shape when the borders of each are the same size. You should experiment with this situation before committing to using shapes and text blocks in this type of configuration, because the exact results are unpredictable.

For more information about the indended use of text blocks, along with code examples that demonstrate the indended use, refer to the discussions of the objects listed under Related Topics.

Properties, Methods, and Events

All of the properties, methods, and events for the BlockFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
FillFormat		
HorizontalAlignment		
LineFormat		
LineSpacing		
LineSpacingPoints		
Opaque		
Orientation		
Parent		
TabWidth		
VerticalAlignment		

Related Topics

TextBlock object
ChildTextBlock object
Department object
HeaderFooter object
TextGraphicObject object
iGrafx API Object Hierarchy

FillFormat Property

Syntax *BlockFormat*.**FillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The FillFormat property returns the FillFormat object for the specified BlockFormat object. The FillFormat object controls whether a fill is used, and if so, what type of fill (solid, pattern, or gradient), and the color or colors used.

For codes examples, refer to the documentation of the FillFormat object.

See Also FillFormat object

iGrafx API Object Hierarchy

{button BlockFormat object,JI(`igrafxrf.HLP',`BlockFormat_Object')}

HorizontalAlignment Property

Syntax *BlockFormat*.**HorizontalAlignment**

Data Type IxHorizontalAlignment enumerated constant (read/write)

Description The HorizontalAlignment property specifies the type of horizontal alignment to use for a particular text block. To align individual lines of text differently, use Paragraph objects for each line of text.

The IxHorizontalAlignment constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixHorizontalAlignLeft
1	ixHorizontalAlignRight
2	ixHorizontalAlignCenter

Example The following example creates a shape in the active diagram. It then adds some text to the shape and using the BlockFormat object, sets the horizontal alignment of the text to be justified right.

```
' Dimension the variables
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text in the shape
igxShape.Text = "A string of text"
MsgBox "View the diagram"
' Set the horizontal alignment to Right
igxShape.TextBlock.BlockFormat.HorizontalAlignment = _
    ixHorizontalAlignRight
MsgBox "View the diagram"
```

See Also Paragraph object

TextBlock object

Shape.Text property

{button BlockFormat object,JI(`igrafxrf.HLP',`BlockFormat_Object')}

LineFormat Property

Syntax *BlockFormat*.**LineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The LineFormat property returns the LineFormat object that is associated with the BlockFormat object. This property allows you to change all of the line formatting attributes of a text block, such as color, style, and width.

Example For code examples, refer to the documentation of the LineFormat object.

See Also LineFormat object

iGrafx API Object Hierarchy

{button BlockFormat object,JI(`igrafxrf.HLP',`BlockFormat_Object')}

LineSpacing Property

Syntax *BlockFormat*.**LineSpacing**

Data Type Double (read/write)

Description The LineSpacing property specifies how much space, in units of lines, to leave between lines of text. Spacing between lines is also referred to as 'leading'. You can use the BlockFormat.LineSpacing property to set line spacing for all paragraphs in the text block to the same amount (compare this to using the ParagraphFormat.LineSpacing property).

To set the spacing in points, use the LineSpacingPoints property.

Note that iGrafx Professional does not provide a method of setting the spacing between paragraphs. To add more space between paragraphs, insert a blank paragraph.

Example The following example creates a shape with text that is set in two paragraphs. It then sets the BlockFormat object's line spacing to three lines, followed by changed the line spacing of the first paragraph to 6 lines with the ParagraphFormat.LineSpacing property.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlk As TextBlock
Dim igxPara As Paragraph
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text of the shape
igxShape.TextLF = "This is Line1, and it is a longer line " _
    & "than most. It actually has a couple of sentences. " _
    & "This will show off a few properties." & Chr(13) _
    & "This is Line2"
MsgBox "View the state of the diagram"
' Get the TextBlock from the Shape object
Set igxTextBlk = igxShape.TextBlock
' Set the line spacing for the text block to 3 lines
igxTextBlk.BlockFormat.LineSpacing = 3
MsgBox "View the state of the diagram. There are " _
    & igxTextBlk.Paragraphs.Count & " paragraphs."
' Show the interaction of setting the line spacing through the
' BlockFormat or ParagraphFormat objects
Set igxPara = igxTextBlk.Paragraphs.Item(1)
igxPara.ParagraphFormat.LineSpacing = 6
MsgBox "View the state of the diagram"
```

See Also LineSpacingPoints property

ParagraphFormat object

{button BlockFormat object,JI(`igrafxrf.HLP','BlockFormat_Object')}

LineSpacingPoints Property

Syntax *BlockFormat*.**LineSpacingPoints**

Data Type Integer (read/write)

Description The LineSpacingPoints property specifies how much space, in units of points, to leave between lines of text. Spacing between lines is also referred to as 'leading'. You can use the BlockFormat.LineSpacingPoints property to set line spacing for all paragraphs in the text block to the same amount (compare this to using the ParagraphFormat.LineSpacingPoints property).

To set the spacing in lines, use the LineSpacing property.

Note that iGrafx Professional does not provide a method of setting the spacing between paragraphs. To add more space between paragraphs, insert a blank paragraph.

Example The following example creates a shape with text, and then sets the line spacing between lines within the paragraph to 20 points with the BlockFormat object, then the spacing for the first paragraph is changed to 36 points using the paragraphFormat object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlk As TextBlock
Dim igxPara As Paragraph
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text of the shape
igxShape.TextLF = "This is Line1, and it is a longer line " _
    & "than most. It actually has a couple of sentences. " _
    & "This will show off a few properties." & Chr(13) _
    & "This is Line2"
MsgBox "View the state of the diagram"
' Get the TextBlock from the Shape object
Set igxTextBlk = igxShape.TextBlock
' Set the line spacing for the text block to 20 points
igxTextBlk.BlockFormat.LineSpacingPoints = 20
MsgBox "View the state of the diagram. There are " _
    & igxTextBlk.Paragraphs.Count & " paragraphs."
' Show the interaction of setting the line spacing through the
' BlockFormat or ParagraphFormat objects
Set igxPara = igxTextBlk.Paragraphs.Item(1)
igxPara.ParagraphFormat.LineSpacingPoints = 36
MsgBox "View the state of the diagram"
```

See Also LineSpacing property

ParagraphFormat object

{button BlockFormat object,JI(`igrafxrf.HLP',`BlockFormat_Object')}

Opaque Property

Syntax *BlockFormat*.**Opaque**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Opaque property specifies whether the block of text is opaque against the background. If the property is set to False, the text is set against the color of the object. If the property is set to True, then the text is 'blocked' against the Windows background color (which can be set by using the Display control panel and the appearances tab).

Using Opaque is desirable if you always want text to be displayed using the color settings you have chosen in the Display control panel.

Example The following example creates a shape in the active diagram. The fill color is set to a solid blue, and then the text block is set to be opaque.

```
' Dimension the variables
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text in the shape
igxShape.Text = "A string of text."
MsgBox "View the diagram"
' Set the properties of the shape
igxShape.FillType = ixFillSolid
igxShape.FillColor = vbBlue
MsgBox "View the diagram"
' Set block format to opaque
igxShape.TextBlock.BlockFormat.Opaque = True
MsgBox "View the diagram"
```

{button BlockFormat object,JI(`igrafxrf.HLP',`BlockFormat_Object')}

Orientation Property

Syntax *BlockFormat*.**Orientation**

Data Type IxOrientation enumerated constant (read/write)

Description The Orientation property allows you to rotate text within a text block. This property rotates the text, not the bounding area for the text. The default value for this property is ixOrientation0, which is standard horizontal text.

The IxOrientation constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixOrientation0
1	ixOrientation90
2	ixOrientation180
3	ixOrientation270

Example The following example creates a text object in the active diagram, and then orients the text so it is rotated 90 degrees.

```
' Dimension the variables
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text in the shape
igxShape.Text = "A string of text."
' Set the Orientation to be rotated 90 degrees
igxShape.TextBlock.BlockFormat.Orientation = ixOrientation90
```

{button BlockFormat object,JI(`igrafxrf.HLP',`BlockFormat_Object')}

TabWidth Property

Syntax *BlockFormat*.**TabWidth**

Data Type Long (read/write)

Description The TabWidth property specifies the tab width value for the BlockFormat object. The units of measure for this property are twips (1440 twips = 1 inch).

Example The following example creates a text block in the active diagram and then set the tab width to ½ inch (720 twips).

```
' Dimension the variables
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text in the shape
igxShape.Text = "A string of text."
MsgBox "View the diagram"
' Set the tab width to ½ inch
igxShape.TextBlock.BlockFormat.TabWidth = 720
igxShape.TextBlock.Paragraphs.Item(1).Indent
MsgBox "View the diagram"
```

{button BlockFormat object,JI(`igrafxrf.HLP',`BlockFormat_Object')}

VerticalAlignment Property

Syntax *BlockFormat*.**VerticalAlignment**

Data Type IxVerticalAlignment enumerated constant (read/write)

Description The VerticalAlignment specifies the type of vertical alignment to use for a particular text block. This property determines the vertical placement of the text within its bounding box.

The IxVerticalAlignment constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixVerticalAlignTop
1	ixVerticalAlignMiddle
2	ixVerticalAlignBottom

Example The following example creates a shape in the active diagram, and adds text to it. Then, using the BlockFormat object, the text is aligned vertically to the bottom of the text block area.

```
' Dimension the variables
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text in the shape
igxShape.Text = "A string of text"
MsgBox "View the diagram"
' Set the vertical alignment to Bottom
igxShape.TextBlock.BlockFormat.VerticalAlignment = _
    ixVerticalAlignBottom
MsgBox "View the diagram"
```

See Also HorizontalAlignment property

{button BlockFormat object,JI(`igrafxrf.HLP',`BlockFormat_Object')}

LinkIndicatorStyle Object

The LinkIndicatorStyle object controls the formatting, or the “style”, of link indicators in a diagram. A link indicator is displayed on shapes that have links to other diagrams, files, or web sites. The link indicator style is set at the diagram level, and affects all shapes within the diagram.

The link indicator can be a small text label (up to three letters long) that is specified by the user and is displayed on each shape that has a link. Alternatively, the link indicator can be either an icon or a drop shadow that is applied to shapes with links.

For more information about the purpose and use of links, refer to the discussions of the Link object and the Links collection object.

The following example shows a typical method of accessing the LinkIndicatorStyle object, which can be accessed only through the Diagram object.

```
' Dimension the variables
Dim igxLinkIndStyle As LinkIndicatorStyle
' Set igxLinkIndStyle variable to the LinkIndicatorStyle object
Set igxLinkIndStyle = ActiveDiagram.LinkIndicatorStyle
```

Properties, Methods, and Events

All of the properties, methods, and events for the LinkIndicatorStyle object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Parent		
Style		
Text		

Related Topics

Link object

Links collection object

iGrafx API Object Hierarchy

Style Property

Syntax *LinkIndicatorStyle*.**Style**

Data Type IxLinkStyle enumerated constant (read/write)

Description The Style property specifies the style of the indicator that is displayed on shapes that contain a link. Each of the link indicator styles replaces the style previously defined; that is, if you intially use the Text style, and then change it to Icon, the link icon displays on the shape and the text does not. Note also that it is possible to set the Text style to an empty string; however, doing so is not all that useful.

The IxLinkStyle constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant	Description
0	ixLinkText	A shape with a link has a text label drawn on it—the text label can be up to three letters and is specified by the Text property of the LinkIndicatorStyle object. For example, you might use the text “L” to indicate a link. 
1	ixLinkShadow	A shape with a link is drawn with a drop shadow. 
2	ixLinkIcon	A shape with a link has a link icon drawn in it’s bottom right corner. 

Note that if you are using shadow formatting for your shapes, you should avoid using the drop shadow style as a link indicator.

Example The following example sets up an initial link indicator of style Icon. A shape is added to the diagram, and then another diagram is added to the document, and a link is created in the shape to this new diagram. The link indicator is then changed to the text style, with the string “LNK”. Now the shape displays the LNK text instead of the icon. Then the link indicator style is specified as a shadow. Finally, the Text property is set to an empty string. Now the shape has no indication that it contains a link.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxLinkIndStyle As LinkIndicatorStyle
Dim igxLink As Link
Dim igxShape As Shape
' Set igxLinkIndStyle variable to the LinkIndicatorStyle object
```

```
Set igxLinkIndStyle = ActiveDiagram.LinkIndicatorStyle
' Set the LinkIndicatorStyle to Icon
igxLinkIndStyle.Style = ixLinkIcon
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text in the shape
igxShape.Text = "A string of text"
' Add a new diagram to the document called NewDiagram
Set igxDiagram = Application.Documents.Item(1).Diagrams.Add _
    ("NewDiagram")
igxDiagram.Views.Item(1).Window.Left = 360
' Add a link on the shape
Set igxLink = igxShape.Links.AddDiagramLink("NewDiagram")
' Set the Description Property to a string
igxLink.Description = "This is a link to another diagram."
MsgBox "View the diagram"
' Set the LinkIndicatorStyle to Text
igxLinkIndStyle.Style = ixLinkText
MsgBox "View the diagram"
igxLinkIndStyle.Text = "LNK"
MsgBox "View the diagram"
' Set the LinkIndicatorStyle to Shadow
igxLinkIndStyle.Style = ixLinkShadow
MsgBox "Link style set to Shadow"
igxLinkIndStyle.Style = ixLinkText
igxLinkIndStyle.Text = ""
MsgBox "Link style Text property set to an empty string."
```

See Also Text Property

NoteIndicatorStyle object

{button LinkIndicatorStyle object,JI(`igrafxrf.HLP',`LinkIndicatorStyle_Object')}

Text Property

Syntax *LinkIndicatorStyle*.**Text**

Data Type String (read/write)

Description The Text property specifies whether a string of text is placed on a shape as the Link indicator. If this property is set to True, the text string is applied to all shapes that have a note associated with them. The text can be a maximum of three characters.

Example The following example sets the link indicator style to text, and sets the text to ‘LNK’.

```
' Dimension the variables
Dim igxShapeFmt As ShapeFormat
Dim igxShape As Shape
Dim igxLinkIndStyle As LinkIndicatorStyle
Dim igxLink As Link
' Set igxLinkIndStyle variable to the LinkIndicatorStyle object.
Set igxLinkIndStyle = ActiveDiagram.LinkIndicatorStyle
' Set LinkIndicatorStyle object to a style constant.
igxLinkIndStyle.Style = ixLinkText
' Set the link indicator text to 'LNK'.
igxLinkIndStyle.Text = "LNK"
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Create a link
Set igxLink = igxShape.Links.AddFileLink _
    ("E:\My Documents\notmyjob.jpg")
MsgBox "View the diagram"
```

See Also Style property

NoteIndicatorStyle object

{button LinkIndicatorStyle object,JI(`igrafxrf.HLP',`LinkIndicatorStyle_Object')}

NoteIndicatorStyle Object

The NoteIndicatorStyle object provides control of Note indicators for all the shapes in a diagram. Note indicators are drawn on shapes that have a Note associated with them. The note indicator style is set at the diagram level, and affects all shapes within the diagram.

Any shape with an attached note always displays the Text property of this object, which defaults to “-N”. You can change this text, or you can also specify to use a shadow as a note indicator.

For more information about the purpose and use of notes, refer to the discussion of the Note object.

The following example shows a typical method of accessing the NoteIndicatorStyle object, which can be accessed only through the Diagram object.

```
' Dimension the variables
Dim igxNoteIndStyle As NoteIndicatorStyle
' Set igxNoteIndStyle variable to the NoteIndicatorStyle object
Set igxNoteIndStyle = ActiveDiagram.NoteIndicatorStyle
```

Properties, Methods, and Events

All of the properties, methods, and events for the NoteIndicatorStyle object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Parent		
Shadow		
Text		

Related Topics

Note object

Shape object

iGrafx API Object Hierarchy

Shadow Property

Syntax *NoteIndicatorStyle*.**Shadow**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Shadow property specifies whether a shadow is applied to all shapes in the specified diagram that have a note associated with them. If a shape does not have a note, then the shadow effect is not applied.

If the shape already has a shadow, the note indicator shadow is not shown. However, if a shape has a note, then the Text property is always displayed (it defaults to “-N”).

Example The following example creates a shape, creates a note for the shape, and sets the diagram's note indicator style to apply a shadow effect to any shape that has a note.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxNoteIndStyle As NoteIndicatorStyle
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Create a Note for the shape
igxShape.Note.Text = "A string of Note text."
' Set igxNoteIndStyle variable to the NoteIndicatorStyle object
Set igxNoteIndStyle = ActiveDiagram.NoteIndicatorStyle
' Set NoteIndicatorStyle object to a style constant
igxNoteIndStyle.Shadow = True
MsgBox "View the diagram"
```

See Also Text property

LinkIndicatorStyle object

{button NoteIndicatorStyle object,JI(`igrafxrf.HLP',`NoteIndicatorStyle_Object')}

Text Property

Syntax *NoteIndicatorStyle*.**Text**

Data Type String (read/write)

Description The Text property specifies a string of text to place at the top, center of all shapes in the specified diagram that have a note associated with them. If a shape does not have a note, then the text specified by this property is not displayed.

The Text property can contain a string up to three characters long. The property defaults to the string, “-N”. Also, this property has no effect on the use of the Shadow property.

Example The following example creates a shape, creates a note for the shape, and sets the diagram's NoteIndicatorStyle.Text property to the text ‘***’.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxNoteIndStyle As NoteIndicatorStyle
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
MsgBox "View the diagram"
' Create a Note for the shape
igxShape.Note.Text = "A string of Note text."
' Set igxNoteIndStyle variable to the NoteIndicatorStyle object
Set igxNoteIndStyle = ActiveDiagram.NoteIndicatorStyle
' Set the note text to ***
igxNoteIndStyle.Text = "***"
MsgBox "View the diagram"
```

See Also Shadow property

LinkIndicatorStyle object

{button NoteIndicatorStyle object,JI(`igrafxrf.HLP',`NoteIndicatorStyle_Object')}

ParagraphFormat Object

The ParagraphFormat object controls the formatting of each paragraph of text in a text block. The following objects use the ParagraphFormat object:

- ChildTextBlock
- Department
- HeaderFooter
- Note
- TextBlock
- TextGraphicObject

For all of the objects listed above, the method of access is as follows:

```
<ParentObj>.Paragraphs.Item(#).Paragraph.ParagraphFormat
```

The TextBlock object (there is only one per shape) and all ChildTextBlock objects (there can be zero or more per shape) have there own distinct ParagraphFormat objects for controlling paragraph text formatting.

The ParagraphFormat object can be assigned to another ParagraphFormat object as a whole. That is, the following assignment statement is valid where Para1 and Para2 are set to valid Paragraph objects.

```
Set Para2.ParagraphFormat = Para1.ParagraphFormat
```

The following example shows a typical method of accessing the ParagraphFormat object.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDiagramObj As DiagramObject
Dim igxTextBlk As TextBlock
Dim igxPara As Paragraph
Dim igxParaFmt As ParagraphFormat
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Set igxDiagramObj variable to the DiagramObject object
Set igxDiagramObj = igxDiagram.DiagramObjects.Item(1)
' Set igxTextBlk variable to a TextBlock object
Set igxTextBlk = igxDiagramObj.Shape.TextBlock
' Set igxPara variable to a Paragraph object
Set igxPara = igxTextBlk.Paragraphs.Item(1)
' Set igxParaFmt variable to a ParagraphFormat object
Set igxParaFmt = igxPara.ParagraphFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the ParagraphFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Alignment		

Application
BulletType
LineSpacing
LineSpacingPoints
Parent

Related Topics

TextBlock object
ChildTextBlock object
iGrafx API Object Hierarchy

Alignment Property

Syntax *ParagraphFormat*.**Alignment**

Data Type IxHorizontalAlignment enumerated constant (read/write)

Description The Alignment property specifies the horizontal alignment of a paragraph. Since paragraphs can be used inside of other objects such as text blocks and child text blocks, this property provides a way to control the alignment of specific paragraphs within a block of text. For example, you could set the text alignment of a text block to 'Left', but set one or more paragraphs within that block to be centered using this property.

The IxHorizontalAlignment constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixHorizontalAlignLeft
1	ixHorizontalAlignRight
2	ixHorizontalAlignCenter

Example The following example creates a shape with two paragraphs of text in it. It then sets the alignment of the first paragraph to be flush with the right side of the shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlk As TextBlock
Dim igxPara As Paragraph
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text of the shape
igxShape.TextLF = "This is a test of the right alignment." _
    & vbCr & "This is a second paragraph without right alignment."
MsgBox "View the diagram"
' Get the TextBlock from the TextGraphicObject
Set igxTextBlk = igxShape.TextBlock
' Set igxPara variable to the first Paragraph object
Set igxPara = igxTextBlk.Paragraphs.Item(1)
' Set the first paragraph to align right
igxPara.ParagraphFormat.Alignment = ixHorizontalAlignRight
MsgBox "View the diagram"
```

See Also BlockFormat object

TextBlock object

ChildTextBlock object

{button ParagraphFormat object,JI(`igrafxrf.HLP',`ParagraphFormat_Object')}

BulletType Property

Syntax *ParagraphFormat*.**BulletType**

Data Type IxBulletType enumerated constant (read/write)

Description The BulletType property allows you to create bulleted lists by specifying a bullet for a given paragraph.

If you format for a bulleted list (any value other than ixBulletNone), you have six choices of bullet styles. Since format objects can be assigned as a whole, it is possible to construct various paragraph format types ahead of time as templates. For instance, you could develop a standard code module to insert into any project that codes for various format objects.

The IxBulletType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixBulletNone
119	ixBulletDiamond
159	ixBulletCircle
167	ixBulletSquare
216	ixBulletArrow
251	ixBulletFancyX
252	ixBulletCheck

Example The following example creates a shape with two paragraphs. It sets the first paragraph to have a bullet of type ixBulletArrow. It sets the second paragraph to have a bullet of type ixBulletCheck.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlk As TextBlock
Dim igxPara As Paragraph
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text of the shape
igxShape.TextLF = "This is Line1" + Chr$(13) + "This is Line2"
' Get the TextBlock from the shape
Set igxTextBlk = igxShape.TextBlock
' Set igxPara variable to the first Paragraph object
Set igxPara = igxTextBlk.Paragraphs.Item(1)
' Set bullet for the text to be an arrow
igxPara.ParagraphFormat.BulletType = ixBulletArrow
MsgBox "View the diagram"
' Set igxPara variable to the second Paragraph object
Set igxPara = igxTextBlk.Paragraphs.Item(2)
' Set bullet for the text to be a check
igxPara.ParagraphFormat.BulletType = ixBulletCheck
MsgBox "View the diagram"
```

{button ParagraphFormat object,JI(`igrafxrf.HLP',`ParagraphFormat_Object')}

LineSpacing Property

Syntax *ParagraphFormat*.**LineSpacing**

Data Type Double (read/write)

Description The LineSpacing property specifies how much space, in units of lines, to leave between lines of text in the same paragraph. Spacing between lines is also referred to as 'leading'. To set the spacing in points, use the LineSpacingPoints property.

Note that iGrafx Professional does not provide a method of setting the spacing between paragraphs. To add more space between paragraphs, insert a blank paragraph.

Example The following example creates a shape with text, and then sets the line spacing between lines within the paragraph to 2.0 lines.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlk As TextBlock
Dim igxPara As Paragraph
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text of the shape
igxShape.TextLF = "This is Line1" & Chr$(13) & "This is Line2"
MsgBox "View the diagram"
' Get the TextBlock from the TextGraphicObject
Set igxTextBlk = igxShape.TextBlock
' Set igxPara variable to the first Paragraph object
Set igxPara = igxTextBlk.Paragraphs.Item(1)
' Set the line spacing to 2.0 lines
igxPara.ParagraphFormat.LineSpacing = 2.0
MsgBox "View the diagram"
```

See Also LineSpacingPoints property

{button ParagraphFormat object,JI(`igrafxrf.HLP',`ParagraphFormat_Object')}

LineSpacingPoints Property

Syntax *ParagraphFormat*.**LineSpacingPoints**

Data Type Integer (read/write)

Description The LineSpacingPoints property specifies how much space, in units of points, to leave between lines of text in the same paragraph. Spacing between lines is also referred to as 'leading'. To set the spacing in lines, use the LineSpacing property.

Note that iGrafx Professional does not provide a method of setting the spacing between paragraphs. To add more space between paragraphs, insert a blank paragraph.

Example The following example creates a shape with text, and then sets the line spacing between lines within the paragraph to 20 points.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxTextBlk As TextBlock
Dim igxPara As Paragraph
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the text of the shape
igxShape.TextLF = "This is Line1" & Chr$(13) & "This is Line2"
MsgBox "View the diagram"
' Get the TextBlock from the TextGraphicObject
Set igxTextBlk = igxShape.TextBlock
' Set igxPara variable to the first Paragraph object
Set igxPara = igxTextBlk.Paragraphs.Item(1)
' Set the line spacing to 20 points
igxPara.ParagraphFormat.LineSpacingPoints = 20
MsgBox "View the diagram"
```

See Also LineSpacing property

{button ParagraphFormat object,JI(`igrafxrf.HLP',`ParagraphFormat_Object')}

OffPageConnectorFormat Object

The OffPageConnectorFormat object controls the formatting of off-page connectors. An off-page connector is a special identifying symbol attached to the departing and arriving connector lines of two connected shapes, as shown in the following illustration. These special symbols are most often used when shapes on different pages are connected. However, it is also possible to use off page connectors for shapes that are on the same page.

Off-Page connector formatting is set at the Diagram level; therefore, all of the off-page connectors within the same diagram use the same format.

The exception is the UseConnectors property of the ConnectorLine object (you can use off page connectors on a line by line basis).

If off page connectors are used, the connector line routing type is always changed to RightAngle, no matter what type is was specified as before setting up the use of off page connectors. You can verify this by reading the Routing property once you change a connector to an off page connector.

The following example shows a typical method of accessing the OffPageConnectorFormat object.

```
' Dimension the variables
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Set igxOffPageConnFmt variable to the OffPageConnectorFormat object
Set igxOffPageConnFmt = ActiveDiagram.OffPageConnectorFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the OffPageConnectorFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
AutomaticConnectors		
DirectionalSymbols		
FillFormat		
Font		
IncludePageNumbers		
LineFormat		
Parent		
ReferenceNumbering		
SharedDestinationConnector		
ToPageFont		
ViewPageBreaks		

Related Topics

ConnectorLine object

iGrafx API Object Hierarchy

AutomaticConnectors Property

Syntax *OffPageConnectorFormat*.**AutomaticConnectors**[= {**True** | **False**}]

Data Type Boolean (read/write)

Description The AutomaticConnectors property specifies whether off page connectors are used automatically when shapes on different pages of a diagram are connected. If the value of this property is False, then off page connectors are not used automatically. If the value is True, then off page connectors are used automatically.

The AutomaticConnectors property affects all connectors in a diagram. If this property is set to False, you can still force an individual connector line to be an off page connector by setting the UseConnectors property of the ConnectorLine object to True.

Example The following example draws three pairs of connected shapes, the first and third pairs each on a different page and the second pair on the same page. The AutomaticConnectors property is then set to True, causing off page connectors to be used for the first and third shape pairs, but not for the second shape pair.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 8 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the first pair of shapes
igxDiagram.DiagramObjects.AddConnectorLine ixRouteDirect, _
    ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirWest, _
    ixConnectRelativeToShape
' Create a third and fourth shapes in the active diagram so they
' are not on separate pages
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the second pair of shapes
igxDiagram.DiagramObjects.AddConnectorLine ixRouteDirect, _
    ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirWest, _
    ixConnectRelativeToShape
' Create a fifth and sixth shape in the active diagram so they
' are again on separate pages
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 5, Application.ShapeLibraries.Item(1).Item(1))
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440 * 5, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the third pair of shapes
igxDiagram.DiagramObjects.AddConnectorLine ixRouteDirect, _
    ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirWest, _
```

```
    ixConnectRelativeToShape
' Get the OffPageConnectorFormat object
Set igxOffPageConnFmt = igxDiagram.OffPageConnectorFormat
' Set the AutomaticConnectors value to true
igxOffPageConnFmt.AutomaticConnectors = True
```

See Also ConnectorLine.UseConnectors property

{button OffPageConnectorFormat object,JI(`igrafxrf.HLP',`OffPageConnectorFormat_Object')}

LineFormat Property

Syntax *OffPageConnectorFormat*.**LineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The LineFormat property returns a LineFormat object for the specified OffPageConnectorFormat object. This LineFormat object controls the formatting of the border of the graphic used to denote the off page connector. For example, in the following illustration, the border of the off page connector graphic has been changed to a dashed line.

Example The following example creates two shapes in the active diagram, spaced 8 inches apart so that each is one a separate page, and connects them with a “direct” connector line (type of ixRouteDirect). It then sets the OffPageConnectorFormat object’s AutomaticConnectors property to True, which automatically creates off page connectors for shapes that are connected across page boundaries. Finally, it sets the border line of the off page connector symbol to red. All other off page connector format properties use their default values.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxLineFmt As LineFormat
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 8 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
igxDiagram.DiagramObjects.AddConnectorLine ixRouteDirect, _
    ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirWest, _
    ixConnectRelativeToShape
' Get the OffPageConnectorFormat object
Set igxOffPageConnFmt = igxDiagram.OffPageConnectorFormat
' Turn on the off page connectors
igxOffPageConnFmt.AutomaticConnectors = True
' Get the BorderFormat object
Set igxLineFmt = igxOffPageConnFmt.LineFormat
' Set the border color for the off page connector to red
igxLineFmt.Color = vbRed
```

See Also LineFormat object

iGrafx API Object Hierarchy

{button OffPageConnectorFormat object,JI(`igrafxrf.HLP',`OffPageConnectorFormat_Object')}

DirectionalSymbols Property

Syntax *OffPageConnectorFormat*.**DirectionalSymbols**[= {**True** | **False**}]

Data Type Boolean (read/write)

Description The DirectionalSymbols property specifies whether directional symbols are used to represent all off page connectors instead of the standard symbol (a circle).

The following illustration shows the directional symbols.

If you decide to use directional symbols, then all off page connectors will use the directional symbols. You cannot control use of the standard symbol or the directional symbol on a connector line by connector line basis.

Example The following example creates two connected shapes in the active diagram. It then gets the OffPageConnectorFormat object from the active diagram and turns on the automatic off page connectors, and the directional symbols.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create the second shape in the active diagram, 8 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
igxDiagram.DiagramObjects.AddConnectorLine ixRouteDirect, _
    ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirWest, _
    ixConnectRelativeToShape
' Get the OffPageConnectorFormat object
Set igxOffPageConnFmt = igxDiagram.OffPageConnectorFormat
' Turn on the off page connectors
igxOffPageConnFmt.AutomaticConnectors = True
' Turn on the directional symbols
igxOffPageConnFmt.DirectionalSymbols = True
```

{button OffPageConnectorFormat object,JI(`igrafxrf.HLP',`OffPageConnectorFormat_Object')}

FillFormat Property

Syntax *OffPageConnectorFormat*.**FillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The FillFormat property returns a FillFormat object that specifies the fill formatting to use for the off page connector symbols. This fill is used for all of the off page connectors in a diagram. The FillFormat object controls whether a fill is used, and if so, what type of fill (solid, pattern, or gradient), and the color or colors used.

In the following illustration, a gradient fill has been chosen for the off page connector fill format.

Example The following example creates two connected shapes in the active diagram. It then gets the OffPageConnectorFormat object from the active diagram and sets a solid fill of yellow for all off page connector symbols.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxFillFmt As FillFormat
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
(1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 8 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
igxDiagram.DiagramObjects.AddConnectorLine ixRouteDirect, _
    ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , _
    igxShape2, ixDirWest, ixConnectRelativeToShape
' Get the OffPageConnectorFormat object
Set igxOffPageConnFmt = igxDiagram.OffPageConnectorFormat
' Turn on the off page connectors
igxOffPageConnFmt.AutomaticConnectors = True
' Get the FillFormat object and set properties
Set igxFillFmt = igxOffPageConnFmt.FillFormat
igxFillFmt.FillType = ixFillSolid
igxFillFmt.FillColor = vbYellow
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button OffPageConnectorFormat object,JI(`igrafxrf.HLP',`OffPageConnectorFormat_Object')}

IncludePageNumbers Property

Syntax *OffPageConnectorFormat*.**IncludePageNumbers**[= {**True | False**}]

Data Type Boolean (read/write)

Description The IncludePageNumbers property specifies whether page numbers are noted below off page connector symbols. The following illustration shows the effect of setting IncludePageNumbers to True.

Example The following example creates two connected shapes in the active diagram. It then gets the OffPageConnectorFormat object, sets AutomaticConnectors to True, and sets IncludePageNumbers to True.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 10 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
igxDiagram.DiagramObjects.AddConnectorLine ixRouteDirect, _
    ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirWest, _
    ixConnectRelativeToShape
' Get the OffPageConnectorFormat object
Set igxOffPageConnFmt = igxDiagram.OffPageConnectorFormat
' Turn on the off page connectors
igxOffPageConnFmt.AutomaticConnectors = True
' Set IncludePageNumbers to True
igxOffPageConnFmt.IncludePageNumbers = True
```

{button OffPageConnectorFormat object,JI(`igrafxrf.HLP',`OffPageConnectorFormat_Object')}

ReferenceNumbering Property

Syntax *OffPageConnectorFormat*.**ReferenceNumbering**

Data Type IxReferenceNumbering enumerated constant (read/write)

Description The ReferenceNumbering property specifies whether the off page connectors are assigned reference numbers or letters. Numbering begins with 1, and lettering begins with A.

The IxReferenceNumbering constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixReferenceNumeric
1	ixReferenceAlphabetic

Example The following example creates two connected shapes in the active diagram. It then gets the OffPageConnectorFormat object, and sets AutomaticConnectors to True. Finally, it sets ReferenceNumbering to ixReferenceNumeric and then to ixReferenceAlphabetic, using a message box to allow the change to be seen..

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 10 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
ActiveDiagram.DiagramObjects.AddConnectorLine ixRouteDirect, _
    ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirWest, _
    ixConnectRelativeToShape
' Get the OffPageConnectorFormat object
Set igxOffPageConnFmt = igxDiagram.OffPageConnectorFormat
' Turn on the off page connectors
igxOffPageConnFmt.AutomaticConnectors = True
' Set ReferenceNumbering to ixReferenceNumeric
igxOffPageConnFmt.ReferenceNumbering = ixReferenceNumeric
MsgBox "View the diagram"
' Set ReferenceNumbering to ixReferenceAlphabetic
igxOffPageConnFmt.ReferenceNumbering = ixReferenceAlphabetic
MsgBox "View the diagram"
```

{button OffPageConnectorFormat object,JI(`igrafxrf.HLP',`OffPageConnectorFormat_Object')}

SharedDestinationConnector Property

Syntax *OffPageConnectorFormat*.**SharedDestinationConnector**[= {**True** | **False**}]

Data Type Boolean (read/write)

Description The SharedDestinationConnector specifies whether an off page connector that is added to a shape is labeled the same as all other connectors that are connected to the shape. If this value is True, then all connectors leading to the shape have the same label. If this value is False, then all connectors leading into the shape are labeled differently.

Example The following example creates two connected shapes in the active diagram. It then gets the OffPageConnectorFormat object, sets AutomaticConnectors to True, and sets SharedDestinationConnector to True, and then back to False.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnLine As ConnectorLine
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1), True)
' Create a second shape in the active diagram, 10 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440, _
    Application.ShapeLibraries.Item(1).Item(1), True)
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
     (ixRouteOrgChart, ixRouteFlagFindEdge, igxShape1, _
     ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
     ixDirWest, ixConnectRelativeToShape)
' Create a third shape in the active diagram
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 4, _
    Application.ShapeLibraries.Item(1).Item(1), True)
' Draw a connector line between the third and second shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
     (ixRouteOrgChart, ixRouteFlagFindEdge, igxShape3, _
     ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
     ixDirSouth, ixConnectRelativeToShape)
MsgBox "View the diagram"
' Get the OffPageConnectorFormat object
Set igxOffPageConnFmt = ActiveDiagram.OffPageConnectorFormat
' Turn on the off page connectors
igxOffPageConnFmt.AutomaticConnectors = True
MsgBox "View the diagram"
' Set the SharedDestinationConnector property to true
igxOffPageConnFmt.SharedDestinationConnector = True
MsgBox "View the diagram"
' Set the SharedDestinationConnector property to false
igxOffPageConnFmt.SharedDestinationConnector = False
MsgBox "View the diagram"
```

{button OffPageConnectorFormat object,JI(`igrafxrf.HLP',`OffPageConnectorFormat_Object')}

ToPageFont Property

Syntax *OffPageConnectorFormat*.**ToPageFont**

Data Type Font object (read-only, See Object Properties)

Description The ToPageFont property returns a Font object for the specified OffPageConnectorFormat object. This Font object specifies the font to use for the “To Page” indicator, which is displayed when the IncludePageNumbers property is set to True. If IncludePageNumbers is set to False, this property has no effect.

Note that the default font size for this property is 6 points. At 100% Zoom factor, a 6 point font may not show the effect of attributes such as Bold or Italic. Zooming to 200% (or using a larger font size) allows you to see these effects.

Example The following example creates three connected shapes in the active diagram. It then gets the OffPageConnectorFormat object, sets AutomaticConnectors to True, and sets IncludePageNumbers to True, and sets the ToPageFont to a, bold, serifed font (Times New Roman).

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnLine As ConnectorLine
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, _
    Application.ShapeLibraries.Item(1).Item(1), True)
' Create a second shape in the active diagram, 10 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440, _
    Application.ShapeLibraries.Item(1).Item(1), True)
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
     (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
     ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
     ixDirWest, ixConnectRelativeToShape)
' Create a third shape in the active diagram
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 4, _
    Application.ShapeLibraries.Item(1).Item(1), True)
' Draw a connector line between the third and second shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
     (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
     ixDirEast, ixConnectRelativeToShape, , , igxShape3, _
     ixDirSouth, ixConnectRelativeToShape)
' Get the OffPageConnectorFormat object
Set igxOffPageConnFmt = ActiveDiagram.OffPageConnectorFormat
' Turn on the OffPage connectors and include the page numbers
igxOffPageConnFmt.AutomaticConnectors = True
igxOffPageConnFmt.IncludePageNumbers = True
MsgBox "View the state of the diagram"
' Set OffPageConnectorFormat ToPageFont properties
igxOffPageConnFmt.ToPageFont.Name = "Arial"
```

```
MsgBox "View the diagram"
igxOffPageConnFmt.ToPageFont.Bold = True
MsgBox "View the diagram"
```

See Also

Font object

iGrafx API Object Hierarchy

{button OffPageConnectorFormat object,JI(`igrafxrf.HLP',`OffPageConnectorFormat_Object')}

ViewPageBreaks Property

Syntax *OffPageConnectorFormat*.**ViewPageBreaks**[= {**True | False**}]

Data Type Boolean (read/write)

Description The ViewPageBreaks property specifies whether page break lines are drawn to indicate the borders of each page in the diagram. Setting the property to True shows the page break lines. Setting the property to False hides the page break lines.

Example The following example creates two connected shapes in the active diagram. It then gets the OffPageConnectorFormat object, sets AutomaticConnectors to True, and sets the off page connector symbol fill to solid yellow. It then toggles the display of the page break line on and off.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine As ConnectorLine
Dim igxFillFmt As FillFormat
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 8 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 8, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Get the OffPageConnectorFormat object
Set igxOffPageConnFmt = ActiveDiagram.OffPageConnectorFormat
' Turn on the off page connectors
igxOffPageConnFmt.AutomaticConnectors = True
' Get the FillFormat object and set properties
Set igxFillFmt = igxOffPageConnFmt.FillFormat
igxFillFmt.FillType = ixFillSolid
igxFillFmt.FillColor = vbYellow
MsgBox "View the state of the diagram"
' Turn off the display of page breaks
igxOffPageConnFmt.ViewPageBreaks = False
MsgBox "View the diagram"
' Turn on the display of page breaks
igxOffPageConnFmt.ViewPageBreaks = True
MsgBox "View the diagram"
```

{button OffPageConnectorFormat object,JI(`igrafxrf.HLP',`OffPageConnectorFormat_Object')}

ArrowFormat Object

The ArrowFormat object controls the formatting of arrowheads on lines. Arrow formatting consists of such properties as color, size, and style (what the arrowhead looks like). Currently, there are 55 styles of arrowhead to choose from.

Arrow formatting is used by the following objects:

- ConnectorLine
- TextGraphicObject (for callout lines)

The Document object also allows you to establish default arrow formats for both source and destination ends of a line. The line can be either a connector line or a callout line.

The following example shows a typical method of accessing the ArrowFormat object. This example assumes that there are at three existing DiagramObject objects in the active diagram, and that the third DiagramObject has a connector line attached to it.

```
' Dimension the variables
Dim igxDiagramObj As DiagramObject
Dim igxConnFmt As ConnectorFormat
Dim igxArrowFmt As ArrowFormat
' Set igxDiagramObj variable to the DiagramObject object
Set igxDiagramObj = ActiveDiagram.DiagramObjects.Item(3)
' Set igxConnFmt variable to the ConnectorFormat object
Set igxConnFmt = igxDiagramObj.ConnectorLine.ConnectorFormat
' Set igxArrowFmt variable to the SourceArrowFormat object
Set igxArrowFmt = igxConnFmt.SourceArrowFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the ArrowFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Color		
Parent		
Size		
Style		

Related Topics

ConnectorLine object

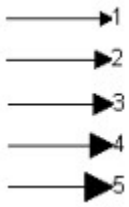
iGrafx API Object Hierarchy

Size Property

Syntax *ArrowFormat*.**Size**

Data Type Integer (read/write)

Description The Size property specifies the size of the arrowhead that is placed on a line. Valid values for this property range from 1 to 5, with 5 being the largest. The following illustration shows the arrowhead sizes.



Example The following example creates two shapes in the active diagram and connects then with a right angle connector line. It then uses two For loops to cycle through the first 5 arrow styles and show each size setting for each style.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine As ConnectorLine
Dim igxConnFmt As ConnectorFormat
Dim igxArrowFmt As ArrowFormat
Dim iCount As Integer
Dim iSize As Integer
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 10 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 5, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set igxConnFmt variable
Set igxConnFmt = igxConnLine.ConnectorFormat
' Set igxArrowFmt variable to access the source arrow properties
Set igxArrowFmt = igxConnFmt.SourceArrowFormat
' Set the source arrow style to None, ixArrowNone
igxArrowFmt.Style = ixArrowNone
MsgBox "View the state of the diagram"
For iCount = 1 To 5
    ' Set the source arrow Style Property using the integer
    ' constants rather than the enumerated constants
    igxArrowFmt.Style = iCount
    For iSize = 1 To 5
        ' Set Size property to each of the valid values
        igxArrowFmt.Size = iSize
```

```
            MsgBox "View the state of the diagram"
        Next iSize
    Next iCount
```

{button ArrowFormat object,JI(`igrafxrf.HLP',`ArrowFormat_Object')}

Style Property

Syntax *ArrowFormat*.**Style**

Data Type IxArrowStyle enumerated constant (read/write)

Description The Style property specifies the type of arrowhead to use. There are currently 55 different arrowhead styles to choose from. To see each of the styles, do either of the following from the user interface:

- Select the connector line and click the right mouse button. Choose the Format option from the popup menu to display the Format Line dialog. Go to the Arrow and Crossovers tab.
- Select the connector line and then choose Format—Lines and Borders from the menu bar to display the Format Line dialog. Go to the Arrow and Crossovers tab.

If this property is set to ixArrowNone, all other properties of this object are ignored.

The IxArrowStyle constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixArrowNone
1-55	IxArrow1-55

Example The following example creates two shapes in the active diagram and connects then with a right angle connector line. It then access the ArrowFormat.Style property to set the source arrow type to ixArrow9.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine As ConnectorLine
Dim igxConnFmt As ConnectorFormat
Dim igxArrowFmt As ArrowFormat
Dim iCount As Integer
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 10 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 5, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set igxConnFmt variable
Set igxConnFmt = igxConnLine.ConnectorFormat
' Set igxArrowFmt variable
Set igxArrowFmt = igxConnFmt.SourceArrowFormat
' Set the source arrow style to None, ixArrowNone
igxArrowFmt.Style = ixArrowNone
MsgBox "View the state of the diagram"
For iCount = 1 To 10
    ' Set the source arrow Style Property using the integer
```

```
    ' constants rather than the enumerated constants
    igxArrowFmt.Style = iCount
    MsgBox "View the state of the diagram"
Next iCount
```

{button ArrowFormat object,JI(`igrafxrf.HLP',`ArrowFormat_Object')}

ConnectorFormat Object

The ConnectorFormat object controls the formatting of connector lines. This object provides control over such properties as line and arrow styles, and how the diagram is to display connector lines that cross each other.

This object provides one means of formatting a connector line. However, the ConnectorLine object also provides properties that control formatting. The advantage to using the ConnectorFormat object is that its properties can be copied completely from one ConnectorLine object to another, as follows:

```
Set igxConnLine1.ConnectorFormat = igxConnLine2.ConnectorFormat
```

Refer to the ConnectorLine object for additional information.

The following example shows a typical method of accessing the ConnectorFormat object.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine As ConnectorLine
Dim igxConnFmt As ConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 10 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 6, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Get the ConnectorFormat object
Set igxConnFmt = igxConnLine.ConnectorFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the ConnectorFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
CrossOverSize		
CrossOverType		
DestinationArrowFormat		
LineFormat		
Parent		
RepeatDestinationArrow		
SourceArrowFormat		

Related Topics

ConnectorLine object
iGrafx API Object Hierarchy

CrossOverSize Property

Syntax *ConnectorFormat*.**CrossOverSize**

Data Type Integer (read/write)

Description The CrossOverSize property specifies the width of the gap when connector lines cross over each other. Valid values for this property are from 1 to 3, with the value 1 creating the smallest gap width. This property is ignored if the CrossOverType property is set to ixCrossLine.

Example The following example creates four shapes in the active diagram, connecting shapes 1 and 2, and shapes 3 and 4 with right angle connector lines. It then sets the crossover type to Square and the crossover size to 3.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnLine As ConnectorLine
Dim igxConnFmt As ConnectorFormat
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram, 10 inches away
' so it is on a different page
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 6, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Create a third shape in the active diagram
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape on the active diagram, 10 inches away
' so it is on a different page
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 4, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape3, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
    ixDirNorth, ixConnectRelativeToShape)
' Get the ConnectorFormat object
Set igxConnFmt = igxConnLine.ConnectorFormat
' Set the ConnectorFormat properties
igxConnFmt.CrossOverType = ixSquare
igxConnFmt.CrossOverSize = 3
```

See Also CrossOverType property

{button ConnectorFormat object,JI(`igrafxrf.HLP',`ConnectorFormat_Object')}

CrossOverType Property

Syntax *ConnectorFormat*.**CrossOverType**

Data Type IxCrossOverType enumerated constant (read/write)

Description The CrossOverType property specifies the style used for drawing crossover points when connector lines cross each other. How the crossover type is applied when connector lines cross depends on whether a line is above or below the line being crossed. For more information, refer to discussion of the ConnectorLine.CrossOverType property.

The IxCrossOverType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixCrossLine
1	ixOverLine
2	ixBreakLine
3	ixSquare
4	ixTriangle

Example For a code example demonstrating the use of this property, refer to the example for the CrossOverSize property.

See Also CrossOverSize property

{button ConnectorFormat object,JI(`igrafxrf.HLP',`ConnectorFormat_Object')}

DestinationArrowFormat Property

Syntax *ConnectorFormat*.**DestinationArrowFormat**

Data Type ArrowFormat object (read-only, See Object Properties)

Description The DestinationArrowFormat property returns an ArrowFormat object for the specified ConnectorFormat object. The ArrowFormat object controls the arrow formatting on the destination end of a connector line (the shape or diagram object that the connector line points into).

Example The following example creates four shapes in the active diagram, connecting shapes 1 and 2, and shapes 3 and 4 with right angle connector lines. It then sets formatting properties for the source end of the for connector 1, and copies those properties to connector 2.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxConnLine2 As ConnectorLine
Dim igxConnFmt As ConnectorFormat
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 6, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Create a third shape in the active diagram
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Create a fourth shape in the active diagram
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 4, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape3, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
    ixDirNorth, ixConnectRelativeToShape)
MsgBox "View the state of the diagram"
' Get the ConnectorFormat object for connector line 1
Set igxConnFmt = igxConnLine1.ConnectorFormat
' Set the ConnectorFormat, Destination Arrow properties
igxConnFmt.DestinationArrowFormat.Style = ixArrow15
igxConnFmt.DestinationArrowFormat.Size = 3
igxConnFmt.DestinationArrowFormat.Color = vbRed
MsgBox "View the state of the diagram"
' Copy the ConnectorFormat from connector 1 to connector 2
igxConnLine2.ConnectorFormat = igxConnFmt
```

```
MsgBox "View the state of the diagram"
```

See Also ArrowFormat object

iGrafx API Object Hierarchy

{button ConnectorFormat object,JI(`igrafxrf.HLP',`ConnectorFormat_Object')}

LineFormat Property

Syntax *ConnectorFormat*.**LineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The LineFormat property returns a LineFormat object for the specified ConnectorFormat object. The LineFormat object controls the formatting of the connector line; for instance, whether it is solid, dashed, dotted, etc., its color, and its size.

Example The following example creates four shapes in the active diagram, connecting shapes 1 and 2, and shapes 3 and 4 with right angle connector lines. It then sets line formatting properties for connector 1, then copies those properties to connector 2. Finally, connector 2's line formatting properties are changed to differ from those of connector 1.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxConnLine2 As ConnectorLine
Dim igxConnFmt As ConnectorFormat
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 6, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Create a third shape in the active diagram
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Create a fourth shape in the active diagram
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 4, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape3, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
    ixDirNorth, ixConnectRelativeToShape)
MsgBox "View the diagram"
' Get the ConnectorFormat object for connector line 1
Set igxConnFmt = igxConnLine1.ConnectorFormat
' Set the ConnectorFormat, Line Format properties
igxConnFmt.LineFormat.Style = ixLineNormal
igxConnFmt.LineFormat.Width = 3
igxConnFmt.LineFormat.Color = vbRed
MsgBox "View the diagram"
' Copy the ConnectorFormat from connector 1 to connector 2
igxConnLine2.ConnectorFormat = igxConnFmt
```

```
MsgBox "View the diagram"
' Change the line properties for connector 2
Set igxConnFmt = igxConnLine2.ConnectorFormat
igxConnFmt.LineFormat.Style = ixLineDashed
igxConnFmt.LineFormat.Width = 2
igxConnFmt.LineFormat.Color = vbBlue
MsgBox "View the diagram"
```

See Also

LineFormat object

iGrafx API Object Hierarchy

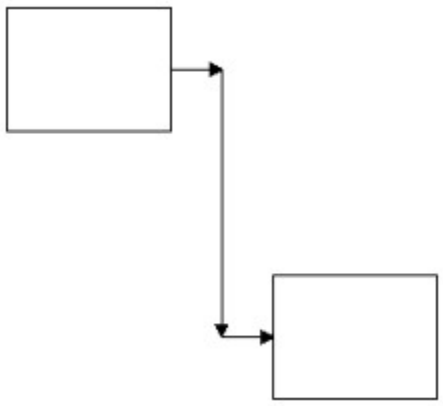
{button ConnectorFormat object,JI(`igrafxrf.HLP',`ConnectorFormat_Object')}

RepeatDestinationArrow Property

Syntax *ConnectorFormat*.**RepeatDestinationArrow**[= {**True** | **False**}]

Data Type Boolean (read/write)

Description The RepeatDestinationArrow specifies whether the destination arrowheads are repeated at the end of every segment of the line. The following illustration shows how connector lines are drawn when this property is set to True.



Example The following example creates four shapes in the active diagram, connecting shapes 1 and 2, and shapes 3 and 4 with right angle connector lines. It then sets line and destination arrow formatting properties for connector 1 and then copies those formats to connector 2. Finally, the RepeatDestinationArrow property is set to True for connector 2.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxConnLine2 As ConnectorLine
Dim igxConnFmt As ConnectorFormat
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 1, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Create a third shape in the active diagram
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Create a fourth shape in the active diagram
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 4, Application.ShapeLibraries.Item(1).Item(1))
```

```
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape3, _
    ixDirWest, ixConnectRelativeToShape, , , igxShape4, _
    ixDirSouth, ixConnectRelativeToShape)
MsgBox "View the diagram"
' Get the ConnectorFormat object for connector line 1
Set igxConnFmt = igxConnLine1.ConnectorFormat
' Set the ConnectorFormat, Line Format properties
igxConnFmt.LineFormat.Style = ixLineNormal
igxConnFmt.LineFormat.Width = 3
igxConnFmt.LineFormat.Color = vbRed
' Set the ConnectorFormat, Destination Arrow properties
igxConnFmt.DestinationArrowFormat.Style = ixArrow5
igxConnFmt.DestinationArrowFormat.Size = 1
igxConnFmt.DestinationArrowFormat.Color = vbGreen
MsgBox "View the diagram"
' Copy the ConnectorFormat from connector 1 to connector 2
igxConnLine2.ConnectorFormat = igxConnFmt
MsgBox "View the diagram"
' Change the line properties for connector 2
Set igxConnFmt = igxConnLine2.ConnectorFormat
igxConnFmt.LineFormat.Style = ixLineDashed
igxConnFmt.LineFormat.Width = 2
igxConnFmt.LineFormat.Color = vbBlue
MsgBox "View the diagram"
' Copy the ConnectorFormat from connector 1 to connector 2
igxConnLine2.ConnectorFormat = igxConnFmt
igxConnLine2.ConnectorFormat.RepeatDestinationArrow = True
MsgBox "View the diagram"
```

SourceArrowFormat Property

Syntax *ConnectorFormat*.**SourceArrowFormat**

Data Type ArrowFormat object (read-only, See Object Properties)

Description The SourceArrowFormat property returns an ArrowFormat object for the specified ConnectorFormat object. The ArrowFormat object controls the arrow formatting on the source end of a connector line (the shape or diagram object that the connector line is leaving).

Example The following example creates four shapes in the active diagram, connecting shapes 1 and 2, and shapes 3 and 4 with connector lines. It then sets formatting properties for the source end of the for connector 1, and copies those properties to connector 2.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxConnLine2 As ConnectorLine
Dim igxConnFmt As ConnectorFormat
Dim igxOffPageConnFmt As OffPageConnectorFormat
' Create the first shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape in the active diagram
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 6, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Create a third shape in the active diagram
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Create a fourth shape in the active diagram
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 4, 1440 * 4, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape3, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
    ixDirNorth, ixConnectRelativeToShape)
MsgBox "View the diagram"
' Get the ConnectorFormat object for connector line 1
Set igxConnFmt = igxConnLine1.ConnectorFormat
' Set the ConnectorFormat, Source Arrow properties
igxConnFmt.SourceArrowFormat.Style = ixArrow22
igxConnFmt.SourceArrowFormat.Size = 3
igxConnFmt.SourceArrowFormat.Color = vbBlue
MsgBox "View the diagram"
' Copy the ConnectorFormat from connector 1 to connector 2
igxConnLine2.ConnectorFormat = igxConnFmt
MsgBox "View the diagram"
```

See Also

ArrowFormat object

iGrafx API Object Hierarchy

{button ConnectorFormat object,JI(`igrafxrf.HLP',`ConnectorFormat_Object')}

FillFormat Object

The FillFormat object controls the formatting of fills for shapes and other types of graphic objects. Fill formatting consists of such properties as the style of fill (solid, gradient, or pattern), the foreground color, and the background color.

The following objects use fill formatting:

- Department object
- Departments object
- Document object
- Graphic object
- ObjectRange object
- Off Page Connectors (through the OffPageConnectorFormat object)
- Shape object (through the ShapeFormat object)
- TextGraphicObject object

The following example shows a typical method of accessing the FillFormat object.

```
' Dimension the variables
Dim igxShapeFmt As ShapeFormat
Dim igxFillFmt As FillFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ShapeFormat object
Set igxShapeFmt = igxShape.ShapeFormat
' Get the FillFormat object
Set igxFillFmt = igxShapeFmt.FillFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the FillFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
BackColor		
FillColor		
FillType		
GradientFormat		
Parent		
PatternIndex		

Related Topics

DefaultFormats object

Department object

Departments object

Graphic object
ObjectRange object
OffPageConnectorFormat object
ShapeFormat object
TextGraphicObject object
iGrafx API Object Hierarchy

BackColor Property

Syntax *FillFormat*.**BackColor**

Data Type Color (read/write)

Description The BackColor property sets the background color of an object. Color values are specified with the RGB function, or with one of the VB color constants.

Certain settings of the FillType and LineStyle properties affect how the background color is used. The following list describes these specific situations.

- If the FillType property is set to IxFillNone or IxFillSolid, the background color has no effect.
- If the FillType property is set to IxFillPattern, the background color is used as the background behind the pattern. Refer to the Format—Fill dialog in the iGrafx Professional user interface, or to the iGrafx Professional User's Guide for more information about fill patterns.
- If the FillType property is set to IxFillGradient, the background color is used as the EndColor in the gradient style (the FillColor is used as the StartColor). Refer to the Format—Fill dialog in the iGrafx Professional user interface, or to the iGrafx Professional User's Guide for more information about gradients.
- If the LineStyle property is set to any of the broken line styles (dashed, dotted, etc.), the background color is used to fill the gaps in the broken line. This allows you to preserve your fill color independent of the lines used for the outline of the graphic.

Example The following example creates a shape on the active diagram, and then fills it with a pattern that is yellow for the FillColor and black for the BackColor.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the FillFormat object through the ShapeFormat object
Set igxFillFmt = igxShape.ShapeFormat.FillFormat
' Set the fill type to pattern, pattern index to 1, back color
' to black and fill color to yellow
igxFillFmt.FillType = ixFillPattern
igxFillFmt.PatternIndex = 1
igxFillFmt.BackColor = vbBlack
igxFillFmt.FillColor = vbYellow
MsgBox "View the diagram"
```

See Also FillType property

{button FillFormat object,JI(`igrafxrf.HLP',`FillFormat_Object')}

FillColor Property

Syntax *FillFormat*.**FillColor**

Data Type Color (read/write)

Description The FillColor property sets the foreground fill color for the specified FillFormat object. It also controls the color of the lines that make up a fill pattern (the background behind a fill pattern is controlled by the BackColor property). Color values are specified with the RGB function, or with one of the Visual Basic color constants.

The value of the FillType property controls how the FillColor property is used.

- If the FillType property is set to IxFillNone, the FillColor property has no effect.
- If the FillType property is set to IxFillSolid, the property specifies the interior fill color. Border lines are not affected by this property.
- If the FillType property is set to IxFillPattern, the FillColor sets the color of the lines that make up the fill pattern. The color of the background behind the pattern lines is controlled by the BackColor property. Refer to the Format—Fill dialog in the iGrafx Professional user interface, or to the iGrafx Professional User's Guide for more information about fill patterns.
- If the FillType property is set to IxFillGradient, the FillColor is used as the StartColor in the gradient style (the BackColor is used as the EndColor). Refer to the Format—Fill dialog in the iGrafx Professional user interface, or to the iGrafx Professional User's Guide for more information about gradients.

Example The following example creates a shape on the active diagram, and then fills it with a pattern that is red for the FillColor and blue for the BackColor.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the FillFormat object through the ShapeFormat object
Set igxFillFmt = igxShape.ShapeFormat.FillFormat
' Set the fill type to pattern, pattern index to 1, back color
' to blue and fill color to red
igxFillFmt.FillType = ixFillPattern
igxFillFmt.PatternIndex = 1
igxFillFmt.BackColor = vbBlue
igxFillFmt.FillColor = vbRed
MsgBox "View the diagram"
```

See Also FillType property

{button FillFormat object,JI(`igrafxrf.HLP',`FillFormat_Object')}

FillType Property

Syntax *FillFormat*.**FillType**

Data Type IxFillType enumerated constant (read/write)

Description The FillType property defines the type of fill to use for the specified object. The FillType property can affect other properties; these effects are described in the table below. For information about using fills with graphics in iGrafx Professional, refer to the iGrafx Professional User's Guide.

The IxFillType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant	Effect on Other Properties
1	ixFillNone	FillColor, GradientIndex, and PatternIndex properties are ignored. BackColor is used to fill in gaps of a line if a broken LineStyle (dashed, dotted, etc) is chosen.
2	ixFillSolid	FillColor sets the foreground interior fill. BackColor is used only if the LineStyle is a broken line. GradientIndex and PatternIndex have no effect.
4	ixFillPattern	FillColor controls the color of the lines that make up the fill pattern. BackColor sets the color behind the pattern lines. PatternIndex sets the pattern to use as the fill. GradientIndex has no effect.
5	ixFillGradient	FillColor is the StartColor of the gradient. BackColor is the EndColor of the gradient. GradientIndex sets the gradient style (type) to use as the fill. PatternIndex has no effect.

Example The following example creates a shape on the active diagram and then fills it with a pattern that is white for the FillColor and black for the BackColor.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the FillFormat object through the ShapeFormat object
Set igxFillFmt = igxShape.ShapeFormat.FillFormat
' Set the fill type to pattern, pattern index to 1, back color
' to black and fill color to white
igxFillFmt.FillType = ixFillPattern
igxFillFmt.PatternIndex = 1
igxFillFmt.BackColor = vbBlack
igxFillFmt.FillColor = vbWhite
MsgBox "View the diagram"
```

{button FillFormat object,JI(`igrafxrf.HLP',`FillFormat_Object')}

GradientFormat Property

Syntax *FillFormat*.**GradientFormat**

Data Type GradientFormat object (read-only, See Object Properties)

Description The GradientFormat property returns the GradientFormat object. This object controls the formatting of gradients when the FillFormat.FillType property is set to ixFillGradient.

Example The following example creates a shape, and then sets the fill to a linear gradient that goes from left to right at a 45 degree angle.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxGradientFmt As GradientFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the FillFormat object through the ShapeFormat object
Set igxFillFmt = igxShape.ShapeFormat.FillFormat
' Set the fill type to gradient
igxFillFmt.FillType = ixFillGradient
' Get the GradientFormat object
Set igxGradientFmt = igxFillFmt.GradientFormat
' Set the gradient formatting properties
igxGradientFmt.Angle = 45
igxGradientFmt.Type = ixGradientLinear
MsgBox "View the diagram"
```

See Also FillType property

GradientFormat object

iGrafx API Object Hierarchy

{button FillFormat object,JI(`igrafxrf.HLP',`FillFormat_Object')}

PatternIndex Property

Syntax *FillFormat*.**PatternIndex**

Data Type Integer (read/write)

Description The PatternIndex property specifies the fill pattern to use as the fill for the specified object. The value can be between 0-32. This property is only valid if the FillType property is set to a value of ixFillPattern.

For this property, the FillColor is used as the color of the lines that make up the pattern, and the BackColor is used as the color behind the pattern of lines. If you need more information about pattern fills, refer to the iGrafx Professional User's Guide.

Example The following example creates a shape on the active diagram, and then fills it with a pattern that is white for the FillColor and black for the BackColor.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the FillFormat object through the ShapeFormat object
Set igxFillFmt = igxShape.ShapeFormat.FillFormat
' Set the fill type to pattern, pattern index to 1, back color
' to black and fill color to white
igxFillFmt.FillType = ixFillPattern
igxFillFmt.PatternIndex = 1
igxFillFmt.BackColor = vbBlack
igxFillFmt.FillColor = vbWhite
MsgBox "View the diagram"
```

See Also FillType property

{button FillFormat object,JI(`igrafxrf.HLP',`FillFormat_Object')}

GradientFormat Object

The GradientFormat object controls the formatting of gradients that are used as fills for shapes and other graphic objects. The GradientFormat object allows you to use one of the builtin gradient styles, or create a custom gradient. This object provides properties for controlling the angle and positioning of the gradient, and the style (linear, radial, or square). The colors used by a gradient are specified by the FillFormat object.

The GradientFormat object is subordinate to the FillFormat object. Therefore, any object that has fill formatting can use gradient formatting. These are:

- Department control
- Departments object
- TextGraphicObject object
- Graphic object
- Shape control (through the ShapeFormat object)
- Off Page Connectors (through the OffPageConnectorFormat object)
- ObjectRange object

The following example shows a typical method of accessing the GradientFormat object.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxGradientFmt As GradientFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the FillFormat object through the ShapeFormat object
Set igxFillFmt = igxShape.ShapeFormat.FillFormat
' Set the fill type to gradient
igxFillFmt.FillType = ixFillGradient
' Get the GradientFormat object
Set igxGradientFmt = igxFillFmt.GradientFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the GradientFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Angle		
Application		
BuiltIn		
Parent		
Type		
XOrigin		
YOrigin		

Related Topics

iGrafx API Object Hierarchy

Angle Property

Syntax *GradientFormat*.**Angle**

Data Type Double (read/write)

Description The Angle property specifies the angle of the gradient. This property is valid when the gradient type is either linear or square, and is ignored when the gradient type is radial.

Example The following example creates a shape, and then set its fill to a linear gradient that goes from left to right at a 45 degree angle.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxGradientFmt As GradientFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the FillFormat object through the ShapeFormat object
Set igxFillFmt = igxShape.ShapeFormat.FillFormat
' Set the fill type to gradient
igxFillFmt.FillType = ixFillGradient
' Get the GradientFormat object
Set igxGradientFmt = igxFillFmt.GradientFormat
' Set the gradient formatting properties
igxGradientFmt.Angle = 45
igxGradientFmt.Type = ixGradientLinear
MsgBox "View the diagram"
```

See Also FillFormat.FillType property

{button GradientFormat object,JI(`igrafxrf.HLP',`GradientFormat_Object')}

BuiltIn Property

Syntax *GradientFormat*.**BuiltIn**[= {**True | False**}]

Data Type Boolean (read-only)

Description The BuiltIn property specifies whether the gradient format for an object matches one of the iGrafx Professional built-in gradient formats. Because users can define their own gradients, this property is useful for determining whether a gradient style is custom or one of the built-in ones provided with iGrafx Professional.

Example The following example gets the first object from the active diagram, and then displays a message box indicating whether or not its gradient format is a built-in.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxGradientFmt As GradientFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.Item(1).Shape
' Get the FillFormat object through the ShapeFormat object
Set igxFillFmt = igxShape.ShapeFormat.FillFormat
' Set the fill type to gradient
igxFillFmt.FillType = ixFillGradient
' Get the GradientFormat object
Set igxGradientFmt = igxFillFmt.GradientFormat
' Test to see if gradient format is a built-in
If (igxGradientFmt.BuiltIn) Then
    MsgBox "Gradient is a built-in."
Else
    MsgBox "Gradient is a not built-in."
End If
```

{button GradientFormat object,JI(`igrafxrf.HLP',`GradientFormat_Object')}

Type Property

Syntax *GradientFormat*.**Type**

Data Type IxGradientType enumerated constant (read/write)

Description The Type property specifies the base type of gradient to use as the fill (linear, radial, or square). Gradient colors are controlled with the FillFormat object. For information about using gradients in iGrafx Professional, refer to the iGrafx Professional User's Guide.

The IxGradientType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixGradientLinear
1	ixGradientRadial
2	ixGradientSquare

If the property's value is set to ixGradientLinear, the XOrigin property has no effect.

Example The following example creates three different shapes on the active diagram, and then applies one of the three different gradient fill types to the shapes.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxGradientFmt As GradientFormat
Dim igxShape As Shape
Dim iCount As Integer
' Loop to create the three different shapes with fills
For iCount = 1 To 3
    ' Create a shape in the active diagram
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * iCount, 1440 * iCount, _
        Application.ShapeLibraries.Item(1).Item(1))
    ' Get the FillFormat object through the ShapeFormat object
    Set igxFillFmt = igxShape.ShapeFormat.FillFormat
    ' Set the fill type of the gradient
    igxFillFmt.FillType = ixFillGradient
    ' Get the GradientFormat object
    Set igxGradientFmt = igxFillFmt.GradientFormat
    ' Set the type of the gradient based on iCount: 1 for linear,
    ' 2 for radial, 3 for square
    Select Case iCount
        Case 1:
            igxGradientFmt.Type = ixGradientLinear
        Case 2:
            igxGradientFmt.Type = ixGradientRadial
        Case 3:
            igxGradientFmt.Type = ixGradientSquare
    End Select
    ' Set the X and Y origins to 50%
    igxGradientFmt.XOrigin = 50
    igxGradientFmt.YOrigin = 50
    MsgBox "View the diagram"
```

```
Next iCount
```

{button GradientFormat object,JI(`igrafxrf.HLP',`GradientFormat_Object')}

XOrigin Property

Syntax *GradientFormat*.**XOrigin**

Data Type Integer (read/write)

Description The XOrigin property specifies the location, horizontally, where the gradient start color begins. The value is given as a percentage (0 to 100). The start color is strongest (undiluted) at the location specified by the XOrigin and YOrigin properties. This property is not valid for linear gradients.

Example The following example creates three different shapes on the active diagram, each filled with a gradient with one of three different XOrigin positions.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxGradientFmt As GradientFormat
Dim igxShape As Shape
Dim iCount As Integer
' Loop to create the three different shapes with fills
For iCount = 1 To 3
    ' Create a shape in the active diagram
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * iCount, 1440 * iCount, _
        Application.ShapeLibraries.Item(1).Item(1))
    ' Get the FillFormat object through the ShapeFormat object
    Set igxFillFmt = igxShape.ShapeFormat.FillFormat
    ' Set the fill type of the gradient
    igxFillFmt.FillType = ixFillGradient
    ' Get the GradientFormat object
    Set igxGradientFmt = igxFillFmt.GradientFormat
    ' Set the gradient formatting properties: Radial, X origin
    ' at 100%, then 50%, then 33%, Y origin at 50%
    igxGradientFmt.Type = ixGradientRadial
    igxGradientFmt.XOrigin = Int(100 / iCount)
    igxGradientFmt.YOrigin = 50
    MsgBox "View the diagram"
Next iCount
```

See Also YOrigin property

{button GradientFormat object,JI(`igrafxrf.HLP',`GradientFormat_Object')}

YOrigin Property

Syntax *GradientFormat*.**YOrigin**

Data Type Integer (read/write)

Description The YOrigin property specifies the location, vertically, where the gradient start color begins. The value is given as a percentage (0 to 100). The start color is strongest (undiluted) at the location specified by the YOrigin and XOrigin properties.

Example The following example creates three different shapes on the active diagram, each filled with a gradient with one of three different YOrigin positions.

```
' Dimension the variables
Dim igxFillFmt As FillFormat
Dim igxGradientFmt As GradientFormat
Dim igxShape As Shape
Dim iCount As Integer
' Loop to create the three different shapes with fills
For iCount = 1 To 3
   ' Create a shape in the active diagram
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * iCount, 1440 * iCount, _
       Application.ShapeLibraries.Item(1).Item(1))
   ' Get the FillFormat object through the ShapeFormat object
   Set igxFillFmt = igxShape.ShapeFormat.FillFormat
   ' Set the fill type of the gradient
   igxFillFmt.FillType = ixFillGradient
   ' Get the GradientFormat object
   Set igxGradientFmt = igxFillFmt.GradientFormat
   ' Set the gradient formatting properties: Radial, X origin
   ' at 50%, Y origin at 100%, then 50%, then 33%
   igxGradientFmt.Type = ixGradientRadial
   igxGradientFmt.XOrigin = 50
   igxGradientFmt.YOrigin = Int(100 / iCount)
   MsgBox "View the diagram"
Next iCount
```

See Also XOrigin property

{button GradientFormat object,JI(`igrafxrf.HLP',`GradientFormat_Object')}

LineFormat Object

The LineFormat object controls the line formatting used to draw the borders of shapes and text blocks, lines of departments, and connector lines. The object provides the same functionality that is available in the user interface through the Line Formatting dialog (Format menu, Lines and Borders option).

The following example shows a typical method of accessing the LineFormat object.

```
' Dimension the variables
Dim igxLineFmt As LineFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the LineFormat object through the ShapeFormat object
Set igxLineFmt = igxShape.ShapeFormat.BorderFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the LineFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Color		
Parent		
Rounding		
Style		
Width		

Related Topics

ConnectorFormat object
Document object
Graphic object
ObjectRange object
TextGraphicObject object
iGrafx API Object Hierarchy

Rounding Property

Topic Under Construction!!!

Syntax *LineFormat*.**Rounding**

Data Type Long (read/write)

Description The Rounding property specifies the amount of rounding to apply to the corners of any???

Example The following example

```
' Dimension the variables
Dim igxShapeFmt As ShapeFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the ShapeFormat object
Set igxShapeFmt = igxShape.ShapeFormat
' Get the LineFormat object
igxShapeFmt.LineFormat.Rounding = 0.3
MsgBox "Line format rounding set to " _
   & igxShapeFmt.LineFormat.Rounding
```

{button LineFormat object,JI(`igrafxrf.HLP',`LineFormat_Object')}

Style Property

Syntax *LineFormat*.**Style**

Data Type IxLineStyle enumerated constant (read/write)

Description The Style property specifies the type of line (solid, dashed, dotted, etc.) used to draw a line. Line formatting applies to departments, connectors, shapes or graphics. The types of line styles can be seen in the user interface by accessing the Format Shape—Line and Border dialog (from the Format menu, choose the Line and Border option, then look at the Line Style drop down list).

The IxLineStyle constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant
-2	ixLineNone
0	ixLineNormal
1	ixLineDashed
2	ixLineDotted
3	ixLineDashDot
4	ixLineDashDotDot

Example The following example creates a shape on the active diagram, and then changes its border line to a dash dot pattern.

```
' Dimension the variables
Dim igxLineFmt As LineFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the LineFormat object and set the style to dash-dot
Set igxLineFmt = igxShape.ShapeFormat.LineFormat
igxLineFmt.Style = ixLineDashDot
MsgBox "View the diagram"
```

{button LineFormat object,JI(`igrafxrf.HLP',`LineFormat_Object')}

ShadowFormat Object

The ShadowFormat object controls shadow effects for various objects. The object provides the same functionality that is available in the user interface through the Shadow Formatting dialog (Format menu, Shadow/3D option).

The following objects use shadow formatting:

- Shape object (through the ShapeFormat object)
- ObjectRange object
- TextGraphicObject object

You can also set a default shadow format for the Document object, which is used when new objects of the types listed above are created.

Various styles of shadow effects are available. Descriptions are provided in the discussion of the Type property. You can see the available option in the Shadow/3D Formatting dialog in the user interface.

The following example shows a typical method of accessing the ShadowFormat object. The ShadowFormat object is accessed through the ShapeFormat object by means of the created shape, and stored in the igxShadowFmt variable.

```
' Dimension the variables
Dim igxShadowFmt As ShadowFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = Application.ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ShadowFormat object through the ShapeFormat object
Set igxShadowFmt = igxShape.ShapeFormat.ShadowFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the ShadowFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Color		
Depth		
Parent		
Type		

Related Topics

DefaultFormats object

ObjectRange object

ShapeFormat object

TextGraphicObject object

iGrafx API Object Hierarchy

Depth Property

Syntax *ShadowFormat*.**Depth**

Data Type Integer (read/write)

Description The Depth property specifies the depth of the shadow effect that is applied to an object. The shadow depth can be a value from 1 to 5. With a value of 1, the shadow depth is 6 points. Each setting increases the depth by 1 point, so that a value of 5 creates a shadow depth of 10 points. This property is ignored if the Type property is set to ixShadowNone (which is the default).

You can assign shadow formatting properties in any order; however, since the default for the Type property is ixShadowNone, you should always explicitly set the type to assure that you get the expected result.

Example The following example gets the ShadowFormat object for the created shape, and stores it in the variable igxShadowFmt. It then sets the depth of the shadow to 5, the maximum amount.

```
' Dimension the variables
Dim igxShadowFmt As ShadowFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ShadowFormat object through the ShapeFormat object
Set igxShadowFmt = igxShape.ShapeFormat.ShadowFormat
' Set the shadow formatting properties
igxShadowFmt.Type = ixShadow1
igxShadowFmt.Color = vbGreen
igxShadowFmt.Depth = 3
MsgBox "View the diagram"
```

{button ShadowFormat object,JI(`igrafxrf.HLP',`ShadowFormat_Object')}

Type Property

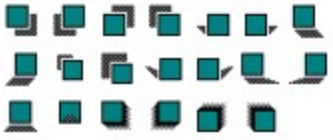
Syntax *ShadowFormat*.**Type**

Data Type IxShadowType enumerated constant (read/write)

Description The Type property specifies the type of shadow effect to apply to an object.

You can assign shadow formatting properties in any order; however, since the default for the Type property is ixShadowNone, you should always explicitly set the type to assure that you get the expected result.

The various shadow effects are shown below:



The IxShadowType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant	Description	
0	ixShadowNone	No shadow.	(Default)
1	ixShadow1		
2	ixShadow2		
3	ixShadow3		
4	ixShadow4		
5	ixShadow5		
6	ixShadow6		
7	ixShadow7		
8	ixShadow8		
9	ixShadow9		
10	ixShadow10		
11	ixShadow11		
12	ixShadow12		
13	ixShadow13		
14	ixShadow14		
15	ixShadow15		
16	ixShadow16		
17	ixShadow17		
18	ixShadow18		
19	ixShadow19		
20	ixShadow20		

You also can view the look of the shadow effects in the user interface by accessing the Format Shape—Shadow/3D dialog (from the Format menu, choose the Shadow/3D option).

Example

The following example gets the ShadowFormat object from the created shape, and stores it in the igxShadowFmt variable. It then changes the style of the shadow to be below and to the right of the object (ixShadow1).

```
' Dimension the variables
Dim igxShadowFmt As ShadowFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ShadowFormat object through the ShapeFormat object
Set igxShadowFmt = igxShape.ShapeFormat.ShadowFormat
' Set the shadow formatting properties
igxShadowFmt.Type = ixShadow1
igxShadowFmt.Color = vbGreen
igxShadowFmt.Depth = 3
MsgBox "View the diagram"
```

{button ShadowFormat object,JI(`igrafxrf.HLP',`ShadowFormat_Object')}

ShapeFormat Object

The ShapeFormat object controls the formatting of Shape objects. Characteristics of a shape controlled by this object include the shape's border, fill, shadowing, and 3D effect. This object provides the same functionality as the following menu options from the user interface: Format—Line and Border, Format—Fill, and Format—Shadow/3D.

The following example shows a typical method of accessing the ShapeFormat object. The ShapeFormat object is accessed from the created Shape object, and stored in the igxShapeFmt variable.

```
' Dimension the variables
Dim igxShapeFmt As ShapeFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = Application.ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ShapeFormat object
Set igxShapeFmt = igxShape.ShapeFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the ShapeFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
FillFormat		
LineFormat		
Parent		
ShadowFormat		
ThreeDFormat		

Related Topics

Shape object

iGrafx API Object Hierarchy

FillFormat Property

Syntax *ShapeFormat*.**FillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The FillFormat property returns a FillFormat object. This object is used to set the fill formatting characteristics for a shape. The FillFormat object controls whether a fill is used, and if so, what type of fill (solid, pattern, or gradient), and the color or colors used.

There are numerous options for fill formats. The example below shows just one of many. Refer to the FillFormat object for more information.

Example The following example creates a shape on the active diagram, and then fills it with a pattern that is white for the FillColor and black for the BackColor.

```
' Dimension the variables
Dim igxShapeFmt As ShapeFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ShapeFormat object
Set igxShapeFmt = igxShape.ShapeFormat
' Access the FillFormat object and set the fill type to pattern,
' pattern index 1, back color of red and fill color of yellow
igxShapeFmt.FillFormat.FillType = ixFillPattern
igxShapeFmt.FillFormat.PatternIndex = 1
igxShapeFmt.FillFormat.BackColor = vbRed
igxShapeFmt.FillFormat.FillColor = vbYellow
MsgBox "View the diagram"
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button ShapeFormat object,JI(`igrafxrf.HLP',`ShapeFormat_Object')}

LineFormat Property

Syntax *ShapeFormat*.**LineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The LineFormat property returns a LineFormat object. This property allows you to change all of the line formatting attributes of the Shape object, such as color, style, and width.

Example The following example creates a shape on the active diagram, and then changes its border to a dash dot pattern.

```
' Dimension the variables
Dim igxShapeFmt As ShapeFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ShapeFormat object
Set igxShapeFmt = igxShape.ShapeFormat
' Set the line formatting properties for the shape
igxShapeFmt.LineFormat.Style = ixLineDashDot
igxShapeFmt.LineFormat.Width = 3
igxShapeFmt.LineFormat.Color = vbGreen
MsgBox "View the diagram"
```

See Also LineFormat object

iGrafx API Object Hierarchy

{button ShapeFormat object,JI(`igrafxrf.HLP',`ShapeFormat_Object')}

ShadowFormat Property

Syntax *ShapeFormat*.**ShadowFormat**

Data Type ShadowFormat object (read-only, See Object Properties)

Description The ShadowFormat property returns a ShadowFormat object. This object is used to set the shadow formatting characteristics for a shape.

For more information about applying shadow effects, refer to the ShadowFormat object.

Example The following example creates a new shape on the active diagram, and then applies shadow formatting to the shape with the ShadowFormat object. The shadow type is set to ixShadow1, the depth to its largest value of 5 (a 10 point shadow), and the color to green.

```
' Dimension the variables
Dim igxShapeFmt As ShapeFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ShapeFormat object
Set igxShapeFmt = igxShape.ShapeFormat
' Set the shadow formatting properties
igxShapeFmt.ShadowFormat.Type = ixShadow1
igxShapeFmt.ShadowFormat.Depth = 5
igxShapeFmt.ShadowFormat.Color = vbGreen
MsgBox "View the diagram"
```

See Also ShadowFormat object

iGrafx API Object Hierarchy

{button ShapeFormat object,JI(`igrafxrf.HLP',`ShapeFormat_Object')}

ThreeDFormat Property

Syntax *ShapeFormat*.**ThreeDFormat**

Data Type ThreeDFormat object (read-only, See Object Properties)

Description The ThreeDFormat property returns a ThreeDFormat object. This object is used to set the three-dimensional formatting characteristics for a shape.

For more information about applying three-dimensional effects, refer to the ThreeDFormat object.

Example The following example creates a shape, and then gets the ThreeDFormat object. It then sets the type and depth of the three dimensional effect.

```
' Dimension the variables
Dim igxShapeFmt As ShapeFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ShapeFormat object
Set igxShapeFmt = igxShape.ShapeFormat
' Set ThreeD formatting properties
igxShapeFmt.ThreeDFormat.Type = ixThreeD1
igxShapeFmt.ThreeDFormat.Depth = 2
MsgBox "View the diagram"
```

See Also ThreeDFormat object

iGrafx API Object Hierarchy

{button ShapeFormat object,JI(`igrafxrf.HLP',`ShapeFormat_Object')}

ThreeDFormat Object

The ThreeDFormat object controls the three-dimensional effects for an object. This formatting makes the object appear to have a solid depth, as if it has volume. The depth of the shape is shaded. The object provides the same functionality that is available in the user interface through the 3D Formatting dialog (Format menu, Shadow/3D option).

The following objects use three-D formatting:

- Shape object (through the ShapeFormat object)
- ObjectRange object
- TextGraphicObject object

You can also set a default 3D format for the Document object, which is used when new objects of the types listed above are created.

Various styles of 3D effects are available. Descriptions are provided in the discussion of the Type property. You can see the available options in the Shadow/3D Formatting dialog in the user interface.

The following example shows a typical method of accessing the ThreeDFormat object. The ThreeDFormat object is accessed from the ShapeFormat object by means of the created shape, and stored in the igxThreeDFmt variable.

```
' Dimension the variables
Dim igxThreeDFmt As ThreeDFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ThreeDFormat object through the ShapeFormat object
Set igxThreeDFmt = igxShape.ShapeFormat.ThreeDFormat
```

Properties, Methods, and Events

All of the properties, methods, and events for the ThreeDFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Depth		
Parent		
Type		

Related Topics

DefaultFormats object
ObjectRange object
ShapeFormat object
TextGraphicObject object
iGrafx API Object Hierarchy

Depth Property

Syntax *ThreeDFormat*.**Depth**

Data Type Integer (read/write)

Description The Depth property specifies the depth of the 3D effect that is applied to an object. The 3D depth can be a value from 1 to 5, with 5 producing the greatest depth. Each setting increases the depth by a predefined amount.

Example The following example creates a shape, and then gets the ThreeDFormat object. It then sets the type and depth of the three dimensional effect.

```
' Dimension the variables
Dim igxThreeDFmt As ThreeDFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ThreeDFormat object through the ShapeFormat object
Set igxThreeDFmt = igxShape.ShapeFormat.ThreeDFormat
' Set the 3D formatting properties
igxThreeDFmt.Type = ixThreeD1
igxThreeDFmt.Depth = 3
MsgBox "View the diagram"
```

{button ThreeDFormat object,JI(`igrafxrf.HLP',`ThreeDFormat_Object')}

Type Property

Syntax *ThreeDFormat*.**Type**

Data Type IxThreeDType enumerated constant (read/write)

Description The Type property specifies the type of 3D effect to apply to an object.

The various 3D effects are shown below:

The IxThreeDType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant	Description
0	ixThreeDNone	No three dimensional effect.
1	ixThreeD1	
2	ixThreeD2	
3	ixThreeD3	
4	ixThreeD4	
5	ixThreeD5	
6	ixThreeD6	
7	ixThreeD7	
8	ixThreeD8	
9	ixThreeD9	
10	ixThreeD10	
11	ixThreeD11	
12	ixThreeD12	
13	ixThreeD13	
14	ixThreeD14	
15	ixThreeD15	
16	ixThreeD16	

The look of the 3D effect can be seen in the user interface by accessing the Format Shape—Shadow/3D dialog (from the Format menu, choose the Shadow/3D option).

Example The following example creates a shape, and then gets the ThreeDFormat object. It then sets the type and depth of the three dimensional effect.

```
' Dimension the variables
Dim igxThreeDFmt As ThreeDFormat
Dim igxShape As Shape
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the ThreeDFormat object through the ShapeFormat object
Set igxThreeDFmt = igxShape.ShapeFormat.ThreeDFormat
```

```
' Set the 3D formatting properties
igxThreeDFmt.Type = ixThreeD1
igxThreeDFmt.Depth = 3
MsgBox "View the diagram"
```

{button ThreeDFormat object,JI(`igrafxrf.HLP',`ThreeDFormat_Object')}

NumberFormat Object

The NumberFormat object controls the textual display of Shape Numbers. The object provides the same functionality that is available in the user interface through the Format dialog for a shape numbering field. All Shape objects have shape numbers.

For example, consider a shape with a three-part hierachical number, say 2.4.1. Depending on the Number Format for the field displaying the number, this could show up as “2-4-1” or “#2.4.1” or “2.4” or “2/4/1/0/0.” The number format acts as a template for displaying the underlying number of the shape.

The following example shows a typical method of accessing the NumberFormat object, which is commonly used for formatting shape numbers.

```
Shape.ShapeNumber.Field.FieldText.NumberFormat
```

Displaying the number on a shape is controlled by the Shape.ShowNumbering property. Other controls related to shape numbers and their display are managed through the Field and FieldText object.

Properties, Methods, and Events

All of the properties, methods, and events for the NumberFormat object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
IncrementingPart		
NumberOfParts		
Parent		
PartType		
Prefix		
Separator		
Suffix		

Related Topics

Field object

Fields object

FieldText object

ShapeNumber object

iGrafx API Object Hierarchy

IncrementingPart Property

Syntax *NumberFormat*.**IncrementingPart**

Data Type Integer (read/write)

Description The IncrementingPart property specifies which part of a shape's number increments.

Important This property's usefulness through automation is limited. Because this property is accessed on a shape-by-shape basis, its only real purpose is for setting up an incrementing part that gets used when a renumber operation is performed through the user interface.

Example The following example illustrates that the IncrementingPart property has no affect on shape numbering when shapes are added to a diagram through automation, and a default shape numbering format has not been established through the user interface.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDiagObj As DiagramObject
Dim iNumParts As Integer
Dim iCount As Integer
' Create the first shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Show shape numbering
igxShape.ShowNumbering = True
' Create a second shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 3, 1440, Application.ShapeLibraries.Item(1).Item(2))
' Show shape numbering
igxShape.ShowNumbering = True
MsgBox "View the diagram"
' Show numbers in hierarchical parts, up to 5, with
' a dot separator
For iCount = 1 To 5
    For Each igxDiagObj In ActiveDiagram.DiagramObjects
        If (igxDiagObj.Type = ixObjectShape) Then
            igxDiagObj.Shape.ShapeNumber.Field.FieldText. _
                NumberFormat.NumberOfParts = iCount
            If (iCount <> 5) Then
                igxDiagObj.Shape.ShapeNumber.Field.FieldText. _
                    NumberFormat.Separator(iCount) = "."
            End If
        End If
    Next igxDiagObj
    MsgBox "View the diagram"
Next iCount
' Set the IncrementingPart to 4
igxShape.ShapeNumber.Field.FieldText.NumberFormat.IncrementingPart = 4
' Add a new shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
' Show shape numbering
igxShape.ShowNumbering = True
MsgBox "View the diagram"
For iCount = 1 To 5
```

```
        With ActiveDiagram.DiagramObjects.Item _
            (ActiveDiagram.DiagramObjects.Count)
            If (.Type = ixObjectShape) Then
                .Shape.ShapeNumber.Field.FieldText. _
                    NumberFormat.NumberOfParts = iCount
                If (iCount <> 5) Then
                    .Shape.ShapeNumber.Field.FieldText. _
                        NumberFormat.Separator(iCount) = "."
                End If
            End If
        End With
        MsgBox "View the diagram"
    Next iCount
```

See Also

NumberOfParts property

Field object

Fields object

FieldText object

ShapeNumber object

{button NumberFormat object,JI(`igrafxrf.HLP',`NumberFormat_Object')}

NumberOfParts Property

Syntax *NumberFormat*.**NumberOfParts**

Data Type Integer (read/write)

Description The NumberOfParts property specifies the number of hierarchical parts of the dhape number to display. For example, to have a numeric field that looks like 4.0.0.0, you would set the value of this property to 4.

To increment any particular part, you would specify that part number with the IncrementingPart property. To set how any particular part of the number is formatted, set the PartType property.

Example The following example creates two shapes in the active diagram and connects them. It then turns on the ShowNumbering property for both shapes. Next, it runs a For loop that increases the number of parts of the shape number that get shown according to the loop index. It then changes the formatting of the third and fourth parts of the number.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iNumParts As Integer
Dim iCount As Integer
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
' Show shape numbering on both shapes
igxShape1.ShowNumbering = True
igxShape2.ShowNumbering = True
MsgBox "View the diagram"
' Show numbers in hierarchical parts, up to 10, with
' a dot separator
For iCount = 1 To 10
   igxShape1.ShapeNumber.Field.FieldText.NumberFormat _
        .NumberOfParts = iCount
    If (iCount <> 10) Then
        igxShape1.ShapeNumber.Field.FieldText.NumberFormat. _
            Separator(iCount) = "."
    End If
    MsgBox "View the diagram"
Next iCount
' Change the PartType to lowercase Alphabetic for part 4, each shape
igxShape1.ShapeNumber.Field.FieldText.NumberFormat _
```

```
    .PartType(4) = ixAlphabeticLower
MsgBox "View the diagram"
' Change the PartType to ZeroPad4 for part 3, each shape
igxShape1.ShapeNumber.Field.FieldText. _
    NumberFormat.PartType(3) = ixNumericZeroPad4
MsgBox "View the diagram"
```

See Also

IncrementingPart property

PartType property

Field object

Fields object

FieldText object

ShapeNumber object

{button NumberFormat object,JI(`igrafxrf.HLP',`NumberFormat_Object')}

PartType Property

Syntax *NumberFormat*.**PartType**(*idx* As Integer)

Data Type IxNumberFormatPartType enumerated constant (read/write)

Description The PartType property specifies the formatting style to use for any particular part of a numeric field. The part of the field to format is specified with the *idx* argument.

NOTE: If the value of a part of the numeric field is zero, and you assign either ixAlphabeticUpper or ixAlphabeticLower to that part, the field does not change from displaying zero as a number. If you use the alphabetic part types, be sure the value of that part is non-zero.

The IxNumberFormatPartType constant defines the valid values for this property, which are listed in the following table. The value ixNumeric is the default.

Value	Name of Constant	Description
0	ixNumeric	The field part uses integer digits: 1, 2, 3, …, etc.
1	ixAlphabeticUpper	The field part uses uppercase alphabetic characters: 1=A, 2=B, 3=C, …, 26=Z, 27=AA, 28=AB, …, etc. The exact sequence of characters may be different depending on which language version of iGrafx Professional you are running.
2	ixAlphabeticLower	The field part uses lowercase alphabetic characters: 1=a, 2=b, 3=c, …, 26=z, 27=aa, 28=ab, …, etc. The exact sequence of characters may be different depending on which language version of iGrafx Professional you are running.
3	ixNumericZeroPad2	The field part uses integer digits and is padded with a zero to the left, if necessary, to assure a two-digit field: 01, 02, 03, …, 10, 11, …, etc.
4	ixNumericZeroPad3	The field part uses integer digits and is padded with up to two zeros to the left, if necessary, to assure a three-digit field: 001, 002, 003, …, 010, 011, …, etc.
5	ixNumericZeroPad4	The field part uses integer digits and is padded with up to three zeros to the left, if necessary, to assure a four-digit field: 0001, 0002, 0003, …, 0010, 0011, …, etc.
6	ixNumericZeroPad5	The field part uses integer digits and is padded with up to four zeros to the left, if necessary, to assure a five-digit field: 00001, 00002, 00003, …, 00010, 00011, …, etc.
7	ixNumericZeroPad6	The field part uses integer digits and is padded with up to five zeros to the left, if necessary, to assure a six-digit field: 000001, 000002, 000003, …, 000010, 000011, …, etc.
8	ixNumericZeroPad7	The field part uses integer digits and is padded with up to six zeros to the left, if necessary, to assure a seven-digit field:

		0000001, 0000002, 0000003, …, 0000010, 0000011, …, etc.
9	ixNumericZeroPad8	The field part uses integer digits and is padded with up to seven zeros to the left, if necessary, to assure a eight-digit field: 00000001, 00000002, 00000003, …, 00000010, 00000011, …, etc.
10	ixNumericZeroPad9	The field part uses integer digits and is padded with up to eight zeros to the left, if necessary, to assure a nine-digit field: 000000001, 000000002, 000000003, …, 000000010, 000000011, …, etc.

Example The following example creates two shapes in the active diagram, and draws a connector line between them. First it sets the PartType of part 1 of the number to each of the possible values. Next, it resets part 1 to ixNumeric. It then sets the display to show four parts, and have a “dot” separator between the parts for Shape 1 and a “dash” separator for Shape 2. Then the part type is changed for parts 3 and 4.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iNumParts As Integer
Dim iCount As Integer
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
' Show shape numbering on both shapes
igxShape1.ShowNumbering = True
igxShape2.ShowNumbering = True
MsgBox "View the diagram"
' Cycle through the different types of formatting for a part
' of a number field for Shape 1
For iCount = 0 To 10
    igxShape1.ShapeNumber.Field.FieldText.NumberFormat _
        .PartType(1) = iCount
    MsgBox "View the diagram"
Next iCount
' Reset the PartType property to ixNumeric
 igxShape1.ShapeNumber.Field.FieldText.NumberFormat _
        .PartType(1) = ixNumeric
```

```
' Display a four-part shape number field for Shape 1
igxShape1.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts = 4
MsgBox "View the diagram"
' Add a dot separator between shape number parts for Shape 1
iNumParts = _
    igxShape1.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts
For iCount = 1 To iNumParts - 1
    igxShape1.ShapeNumber.Field.FieldText.NumberFormat. _
        Separator(iCount) = "."
Next iCount
MsgBox "View the diagram"
' Make Shape 2 number format 4 parts
igxShape2.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts = 4
' Add a dash as the separator between the shape number parts, Shape 2
iNumParts = _
    igxShape2.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts
For iCount = 1 To iNumParts - 1
    igxShape2.ShapeNumber.Field.FieldText.NumberFormat. _
        Separator(iCount) = "-"
Next iCount
MsgBox "View the diagram"
' Change the PartType to lowercase Alphabetic for part 4, each shape
igxShape1.ShapeNumber.Field.FieldText.NumberFormat _
        .PartType(4) = ixAlphabeticLower
igxShape2.ShapeNumber.Field.FieldText.NumberFormat _
        .PartType(4) = ixAlphabeticLower
MsgBox "View state of the diagram"
' Change the PartType to ZeroPad4 for part 3, each shape
igxShape1.ShapeNumber.Field.FieldText. _
    NumberFormat.PartType(3) = ixNumericZeroPad4
igxShape2.ShapeNumber.Field.FieldText. _
    NumberFormat.PartType(3) = ixNumericZeroPad4
MsgBox "View the diagram"
```

See Also

Field object

Fields object

FieldText object

ShapeNumber object

{button NumberFormat object,JI(`igrafxrf.HLP',`NumberFormat_Object')}

Prefix Property

Syntax *NumberFormat*.**Prefix**

Data Type String (read/write)

Description The Prefix property specifies a user-defined string that is prepended to the beginning of a number field. A common number field you might use is the shape number. Shape numbers are hierarchical, and their display can range from a simple one part style, such as 4, to a more complex multi-part style, such as 4.0.a.3, etc. For example, if the prefix string were “number: “ then numbers would be displayed as “number: 2” or “number: 9.”

Example The following example adds a prefix to the shape numbers being displayed on the diagram’s two shapes. The prefix is specified as “Number:”.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iNumParts As Integer
Dim iCount As Integer
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
' Show shape numbering on both shapes
igxShape1.ShowNumbering = True
igxShape2.ShowNumbering = True
MsgBox "View the diagram"
' Display a four-part shape number field for Shape 1
igxShape1.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts = 4
MsgBox "View the diagram"
' Add a dot as the separator between the parts of the shape number
iNumParts = _
   igxShape1.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts
For iCount = 1 To iNumParts - 1
   igxShape1.ShapeNumber.Field.FieldText.NumberFormat. _
       Separator(iCount) = "."
Next iCount
MsgBox "View the diagram"
' Add a prefix to the shape numbers
igxShape1.ShapeNumber.Field.FieldText. _
    NumberFormat.Prefix = "Number:"
igxShape2.ShapeNumber.Field.FieldText. _
    NumberFormat.Prefix = "Number:"
```

```
MsgBox "View the diagram"
```

See Also

Field object

Fields object

FieldText object

ShapeNumber object

{button NumberFormat object,JI(`igrafxrf.HLP',`NumberFormat_Object')}

Separator Property

Syntax *NumberFormat*.**Separator**(*idx* As Integer)

Data Type String (read/write)

Description The Separator property specifies a user-defined string to use as a separator between parts of a numeric field. A common number field you might use is the shape number, but the property can be applied to any numeric field.

The separator always follows the numeric field part, and the *idx* argument allows you to specify the separator for a particular part of the numeric field. Typical values are dashes or periods. A different separator can be specified after each number part.

Numeric fields can have one or more “parts”. Using the shape number as an example, you could have a simple one-part style such as 4, or a multi-part style such as 4.0.a.3. For instance, using the multi-part example, if you wanted to change the separator from a dot to a dash, you would write:

```
Separator(1) = "-"
```

which would result in 4-0.a.3.

Example The following example shows how to specify separators between the parts of a shape number.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iNumParts As Integer
Dim iCount As Integer
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
' Show shape numbering on both shapes
igxShape1.ShowNumbering = True
igxShape2.ShowNumbering = True
MsgBox "View the diagram"
' Display a four-part shape number field for Shape 1
igxShape1.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts = 4
MsgBox "View the diagram"
```

Notice how at this point, even though you have set the number of parts to 4, there is no visual indication that you have a hierarchical number with 4 separate parts. Next, add the following code to the end of the previous code section. You now have a “dot” that separates each part of the number.

```
' Add a dot as the separator between the parts of the shape number
iNumParts = _
   igxShape1.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts
For iCount = 1 To iNumParts - 1
   igxShape1.ShapeNumber.Field.FieldText.NumberFormat. _
      Separator(iCount) = "."
Next iCount
MsgBox "View the diagram"
```

Finally, add this last block of code to format the shape number of Shape 2 with a “dash” as the separator.

```
igxShape2.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts = 4
' Add a dash as the separator between the shape number parts, Shape 2
iNumParts = _
   igxShape2.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts
For iCount = 1 To iNumParts - 1
   igxShape2.ShapeNumber.Field.FieldText.NumberFormat. _
      Separator(iCount) = "-"
Next iCount
MsgBox "View the diagram"
```

See Also

Field object

Fields object

FieldText object

ShapeNumber object

{button NumberFormat object,JI(`igrafxrf.HLP',`NumberFormat_Object')}

Suffix Property

Syntax *NumberFormat*.**Suffix**

Data Type String (read/write)

Description The Suffix property specifies a user-defined string that is appended to the end of a number field. A common number field you might use is the shape number. Shape numbers are hierarchical, and their display can range from a simple one part style, such as 4, to a more complex multi-part style, such as 4.0.a.3, etc. For example, if the prefix string were ":Shape" then numbers would be displayed as "1:Shape", "2:Shape", etc.

Example The following example adds a suffix to the shape numbers being displayed on the diagram's two shapes. The suffix is specified as ":Shape".

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iNumParts As Integer
Dim iCount As Integer
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
' Show shape numbering on both shapes
igxShape1.ShowNumbering = True
igxShape2.ShowNumbering = True
MsgBox "View state of the diagram"
' Display a four-part shape number field for Shape 1
igxShape1.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts = 4
MsgBox "View the diagram"
' Add a dot as the separator between the parts of the shape number
iNumParts = _
   igxShape1.ShapeNumber.Field.FieldText.NumberFormat.NumberOfParts
For iCount = 1 To iNumParts - 1
   igxShape1.ShapeNumber.Field.FieldText.NumberFormat. _
       Separator(iCount) = "."
Next iCount
MsgBox "View the diagram"
' Add a suffix to the shape numbers
igxShape1.ShapeNumber.Field.FieldText. _
    NumberFormat.Suffix = ":Shape"
igxShape2.ShapeNumber.Field.FieldText. _
    NumberFormat.Suffix = ":Shape"
```

```
MsgBox "View the diagram"
```

See Also

Field object

Fields object

FieldText object

ShapeNumber object

{button NumberFormat object,JI(`igrafxrf.HLP',`NumberFormat_Object')}

DefaultFormats Object

Topic Under Construction!!!

The code for setting the DefaultFormats object can be placed in any code project (Application/Extension, Document, or Diagram).

The default formats only affect connector lines that are drawn interactively through the user interface. Lines drawn using VBA are not affected.

Properties, Methods, and Events

All of the properties, methods, and events for the DefaultFormats object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
ConnectorLineFillFormat		
ConnectorLineLineFormat		
ConnectorLineShadowFormat		
ConnectorLineThreeDFormat		
DestinationArrowFormat		
Parent		
ShapeFillFormat		
ShapeLineFormat		
ShapeShadowFormat		
ShapeThreeDFormat		
SourceArrowFormat		

ConnectorLineFillFormat Property

Syntax *DefaultFormats*.**ConnectorLineFillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The ConnectorLineFillFormat property returns a FillFormat object that defines default values for the fill formatting used for any connector line that is added to any diagram in a document. The DefaultFormats object is a property of the Document object. The property is useful for setting up a default fill format for connector lines. One use is when you are constructing templates.

Example The following example creates four shapes in the active diagram to use for drawing connector lines between interactively. The ConnectorLineFillFormat property is used to set fill formatting defaults for “filled” connector lines. Run the code, and then go to the user interface to draw connector lines between the shapes. Select connector lines and set the Filled option in the Format—Lines and Borders dialog tab.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default ConnectorLine formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
     ' Set default fill format
    .ConnectorLineFillFormat.FillType = ixFillSolid
    .ConnectorLineFillFormat.FillColor = vbBlue
    .ConnectorLineFillFormat.BackColor = vbYellow
End With
MsgBox "Return to the interface and draw connectors between " _
    & "the shapes." & Chr(13) & "Next, select a connector and " _
    & " choose the Format--Line and Border menu item" & Chr(13) _
    & "Click the Filled option. The connector uses the " _
    & "established defaults."
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

ConnectorLineLineFormat Property

Syntax *DefaultFormats*.**ConnectorLineLineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The ConnectorLineLineFormat property returns a LineFormat object that defines default values for the line formatting used for any connector line that is added to any diagram in a document. The DefaultFormats object is a property of the Document object. The property is useful for setting up a default line format for connector lines. One use is when you are constructing templates.

Example The following example creates four shapes in the active diagram to use for drawing connector lines between interactively. The ConnectorLineLineFormat property is used to set the line formatting defaults for all connector lines. Run the code, and then go to the user interface to draw connector lines between the shapes. Select connector lines and set the Filled option in the Format—Lines and Borders dialog tab.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default ConnectorLine formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
     ' Set default line format
     .ConnectorLineLineFormat.Style = ixLineDashed
     .ConnectorLineLineFormat.Width = 40
     .ConnectorLineLineFormat.Color = vbCyan
End With
MsgBox "Return to the interface and draw connectors between " _
    & "the shapes." & Chr(13) & "The connector uses the " _
    & "established defaults."
```

See Also LineFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

ConnectorLineShadowFormat Property

Syntax *DefaultFormats*.**ConnectorLineShadowFormat**

Data Type ShadowFormat object (read-only, See Object Properties)

Description The ConnectorLineShadowFormat property returns a ShadowFormat object that defines default values for the shadow formatting used for any “filled” connector line that is added to any diagram in a document. The DefaultFormats object is a property of the Document object. The property is useful for setting up a default shadow format for connector lines. One use is when you are constructing templates.

Note Shadowing only applies to “filled” connector lines. To create a filled connector line, select a connector and check the Filled box in the Format—Lines and Borders dialog.

Example The following example creates four shapes in the active diagram to use for drawing connector lines between interactively. The ConnectorLineShadowFormat property is used to set defaults for “filled” connector lines. Run the code, and then go to the user interface to draw connector lines between the shapes. Select connector lines and set the Filled option in the Format—Lines and Borders dialog tab. Then go to the Shadow/3D tab and click the Shadow button.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default ConnectorLine formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
     ' Set default shadow format
    .ConnectorLineShadowFormat.Type = ixShadow1
    .ConnectorLineShadowFormat.Depth = 2
    .ConnectorLineShadowFormat.Color = vbBlue
End With
MsgBox "Return to the interface and draw connectors between " _
    & "the shapes." & Chr(13) & "Next, select a connector and " _
    & "choose the Format--Line and Border menu item" & Chr(13) _
    & "Click the Filled option. Then go to the Shadow/3D tab " _
    & "and click the Shadow button." & "The connector uses the " _
    & "established defaults."
```

See Also ShadowFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

ConnectorLineThreeDFormat Property

Syntax *DefaultFormats*.**ConnectorLineThreeDFormat**

Data Type ThreeDFormat object (read-only, See Object Properties)

Description The ConnectorLineThreeDFormat property returns a ThreeDFormat object that defines default values for the ThreeD formatting used for any “filled” connector line that is added to any diagram in a document. The DefaultFormats object is a property of the Document object. The property is useful for setting up a default ThreeD format for connector lines. One use is when you are constructing templates.

Note Three-dimensional effects only apply to “filled” connector lines. To create a filled connector line, select a connector and check the Filled box in the Format—Lines and Borders dialog.

Example The following example creates four shapes in the active diagram to use for drawing connector lines between interactively. The ConnectorLineThreeDFormat property is used to set defaults for “filled” connector lines. Run the code, and then go to the user interface to draw connector lines between the shapes. Select connector lines and set the Filled option in the Format—Lines and Borders dialog tab. Then go to the Shadow/3D tab and click the 3D button.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default ConnectorLine formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
   ' Set default 3-D format
   .ConnectorLineThreeDFormat.Type = ixThreeD10
   .ConnectorLineThreeDFormat.Depth = 3
End With
MsgBox "Return to the interface and draw connectors between " _
    & "the shapes." & Chr(13) & "Next, select a connector and " _
    & "choose the Format--Line and Border menu item" & Chr(13) _
    & "Click the Filled option. Then go to the Shadow/3D tab " _
    & "and click the 3D button." & "The connector uses the " _
    & "established defaults."
```

See Also ThreeDFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

DestinationArrowFormat Property

Syntax *DefaultFormats*.**DestinationArrowFormat**

Data Type ArrowFormat object (read-only, See Object Properties)

Description The DestinationArrowFormat property returns an ArrowFormat object that defines default values for the arrow formatting to use for the destination end of any connector line that is added to any diagram in a document. The DefaultFormats object is a property of the Document object. The property is useful for setting up a default arrow format for connector lines. One use is when you are constructing templates.

Example The following example creates four shapes in the active diagram to use for drawing connector lines between interactively. The DestinationArrowFormat property is used to set the formatting defaults for destination arrows for all connector lines. Run the code, and then go to the user interface to draw connector lines between the shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default ConnectorLine formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
    ' Set default destination arrow format
   .DestinationArrowFormat.Style = ixArrow17
   .DestinationArrowFormat.Size = 4
   .DestinationArrowFormat.Color = vbGreen
End With
MsgBox "Return to the interface and draw connectors between " _
    & "the shapes." & Chr(13) & "The connector uses the " _
    & "established defaults."
```

See Also ArrowFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

ShapeFillFormat Property

Syntax *DefaultFormats*.**ShapeFillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The ShapeFillFormat property returns a FillFormat object that defines default values for the fill formatting used for any shape that is added to any diagram in a document through the user interface (not through VBA). The DefaultFormats object is a property of the Document object. The property is useful for setting up a default fill format for shapes. One use is when you are constructing templates.

Example The following example creates four shapes in the active diagram. This shows that the DefaultFormats object does not affect shapes added through VBA. The ShapeFillFormat property is used to set the fill formatting defaults for all shapes added to a diagram interactively. Run the code, and then go to the user interface to draw several shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default Shape formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
    ' Set default shape fill format
   .ShapeFillFormat.FillType = ixFillPattern
   .ShapeFillFormat.PatternIndex = 8
   .ShapeFillFormat.BackColor = vbRed
   .ShapeFillFormat.FillColor = vbYellow
End With
MsgBox "Return to the interface and draw some shapes." & Chr(13) _
    & "The shapes you add use the established defaults."
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

ShapeLineFormat Property

Syntax *DefaultFormats*. **ShapeLineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The ShapeLineFormat property returns a LineFormat object that defines default values for the line formatting used for any shape that is added to any diagram in a document through the user interface (not through VBA). The DefaultFormats object is a property of the Document object. The property is useful for setting up a default line format for shapes. One use is when you are constructing templates.

Example The following example creates four shapes in the active diagram. This shows that the DefaultFormats object does not affect shapes added through VBA. The ShapeLineFormat property is used to set the line formatting defaults for all shapes added to a diagram interactively. Run the code, and then go to the user interface to draw several shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default Shape formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
    ' Set default shape line format
   .ShapeLineFormat.Style = ixLineDashDotDot
   .ShapeLineFormat.Width = 60
   .ShapeLineFormat.Color = vbGreen
End With
MsgBox "Return to the interface and draw some shapes." & Chr(13) _
    & "The shapes you add use the established defaults."
```

See Also LineFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

ShapeShadowFormat Property

Syntax *DefaultFormats*. **ShapeShadowFormat**

Data Type ShadowFormat object (read-only, See Object Properties)

Description The ShapeShadowFormat property returns a ShadowFormat object that defines default values for the shadow formatting used for any shape that is added to any diagram in a document through the user interface (not through VBA). The DefaultFormats object is a property of the Document object. The property is useful for setting up a default shadow format for shapes. One use is when you are constructing templates.

Example The following example creates four shapes in the active diagram. This shows that the DefaultFormats object does not affect shapes added through VBA. The ShapeShadowFormat property is used to set the shadow formatting defaults for all shapes added to a diagram interactively. Run the code, and then go to the user interface to draw several shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default Shape formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
    ' Set default shape shadow format
    .ShapeShadowFormat.Type = ixShadow15
   .ShapeShadowFormat.Depth = 3
   .ShapeShadowFormat.Color = vbRed
End With
MsgBox "Return to the interface and draw some shapes." & Chr(13) _
    & "The shapes you add use the established defaults."
```

See Also ShadowFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

ShapeThreeDFormat Property

Syntax *DefaultFormats*. **ShapeThreeDFormat**

Data Type ThreeDFormat object (read-only, See Object Properties)

Description The ShapeThreeDFormat property returns a ThreeDFormat object that defines default values for the ThreeD formatting used for any shape that is added to any diagram in a document through the user interface (not through VBA). The DefaultFormats object is a property of the Document object. The property is useful for setting up a default ThreeD format for shapes. One use is when you are constructing templates.

Example The following example creates four shapes in the active diagram. This shows that the DefaultFormats object does not affect shapes added through VBA. The ShapeThreeDFormat property is used to set the three-D formatting defaults for all shapes added to a diagram interactively. Run the code, and then go to the user interface to draw several shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default Shape formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
    ' Set default shape 3D format
    .ShapeThreeDFormat.Type = ixThreeD14
    .ShapeShadowFormat.Depth = 4
End With
MsgBox "Return to the interface and draw some shapes." & Chr(13) _
    & "The shapes you add use the established defaults."
```

See Also ThreeDFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

SourceArrowFormat Property

Syntax *DefaultFormats*.**SourceArrowFormat**

Data Type ArrowFormat object (read-only, See Object Properties)

Description The SourceArrowFormat property returns an ArrowFormat object that defines default values for the arrow formatting to use for the source end of any connector line that is added to any diagram in a document. The DefaultFormats object is a property of the Document object. The property is useful for setting up a default arrow format for connector lines. One use is when you are constructing templates.

Example The following example creates four shapes in the active diagram to use for drawing connector lines between interactively. The SourceArrowFormat property is used to set the formatting defaults for source arrows for all connector lines. Run the code, and then go to the user interface to draw connector lines between the shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDocDefaults As DefaultFormats
' Create several shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 4)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440 * 4)
MsgBox "Shapes added to diagram"
' Set the default ConnectorLine formats
Set igxDocDefaults = ActiveDocument.DefaultFormats
With igxDocDefaults
    ' Set default source arrow format
   .SourceArrowFormat.Style = ixArrow10
   .SourceArrowFormat.Size = 4
   .SourceArrowFormat.Color = vbRed
End With
MsgBox "Return to the interface and draw connectors between " _
    & "the shapes." & Chr(13) & "The connector uses the " _
    & "established defaults."
```

See Also ArrowFormat object

iGrafx API Object Hierarchy

{button DefaultFormats object,JI(`igrafxrf.HLP',`DefaultFormats_Object')}

Application Property

Syntax <*Object Name*>.**Application**

Data Type Application object (read-only, See Object Properties)

Description The Application property returns the iGrafx Professional Application object. This is the highest level of the object hierarchy. The purpose of this property is to allow the developer to quickly access the application level of the object model from any other (deeper) level of the object hierarchy. Through the Application property, you can access any of the properties, methods, and events of the Application object.

The only objects in the iGrafx Professional API that do not have an Application property are:

- Change object
- CommandHandler object
- IGrafxExtension object

Refer to the Application object for information about the properties, methods and events.

Caption Property

Syntax <*Object Name*>.**Caption**

Data Type String (read/write)

Description The Caption property specifies a string of text to display. This property is common to several objects in the iGrafx Professional API. These objects, and a description of what the Caption property controls for each object, is presented in the following table. For all objects, the property is read/write, except for the Window object, for which the Caption property is read-only.

Object Name	Description of Caption Property	Example
Application	For the Application object, the Caption property specifies the application name, and is displayed starting at the left edge of the application window's title bar. Other text (such as the Document or Diagram name) may follow the Caption, depending on which application options or preferences are set.	See the example for the Application.Build property.
CommandBarItem	For the CommandBarItem object, the Caption property specifies the name of the item. For example, for File—New, the string "New" is the caption of the command bar item. You cannot change the caption of built-in command bar items, only those you create.	See the example for the CommandBarItem.BuiltIn property.
PercentGauge	For the PercentGauge object, the Caption property specifies the text in the Window Bar of the Percent Gauge window.	See the example for the PercentGauge object.
OutputPane	For the OutputPane object, the Caption property specifies the name of the output pane. The caption is displayed as a Tab at the bottom left of the Output Window. If additional panes are added, they are added as Tabs to the right of any previously created panes.	See the example for the OutputPanes.Item method.
Window	For the Window object, the Caption property is read-only, and allows the programmer to retrieve the name of the window that is displayed in the window's title bar. This name is assigned by the application.	See the example for the Application.Window property.

Count Property

Syntax <*Object Name*>.**Count**

Data Type Long (read-only)

Description The Count property returns the number of items in a collection. All collection objects have a Count property, which allows the programmer to determine how many items are contained in the collection. This property is read-only for all objects.

The syntactical form, and the approach to using the property is the same for all collection objects. The following table lists the objects that have a Count property. The names are linked to the Item method for each object, in which the Example section shows a use of the Count property.

Adjustments	ChildTextBlocks	CommandBarItems
CommandBars	CommandCategories	ComponentRange
Components	ConnectPoints	CustomDataDefinitions
CustomDataValues	DecisionCases	DepartmentNames
DepartmentRange	Departments	DiagramObjects
DiagramRange	Diagrams	DiagramTypes
Documents	Entities	ExcludedDepartmentNames
ExtensionProjects	Fields	FieldTexts
FontNames	GalleryPanes	Graphics
Guidelines	Layers	Links
ObjectRange	OutputPanes	Pages
Paragraphs	Paths	Points
PolygonPoints	PolyPolygonGraphic	PopupWindows
PropertyList	PropertyLists	RecentFiles
ShapeLibraries	ShapeLibrary	StartPointNames
Templates	Views	Windows

Color Property

Syntax *<Object Name>*.**Color**

Data Type Long (read/write)

Description The Color property specifies the color for a particular object. The most common ways to set the value of this property is to use either the Visual Basic RGB function or to use one of the Visual Basic color constants (for example, vbRed).

When you query the color of an object, the value returned is of type Long.

The following objects have a color property:

- ArrowFormat
- Entity
- Font
- LineFormat
- ShadowFormat

Example The following example sets the value of a color property. The syntax is the same for all of the aforementioned objects. In the examples, replace <ObjectName> with the appropriate object name.

```
<Object Name>.Color = RGB(240, 120, 0)

<Object Name>.Color = vbRed
```

Related Other properties that specify or return a color are:

- BackColor
- BorderColor
- DestinationArrowColor
- FillColor
- IntersectionColor
- LineColor
- MaskColor
- ShadowColor
- SourceArrowColor

Font Property

Syntax *<Object Name>*.**Font**

Data Type Font object (read-only, See Object Properties)

Description The Font property returns the Font object of the object specified by <Object Name>. The syntax is the same as for any other object property. Refer to the Font object for more information.

The following objects have a Font property:

- Field object
- Legend object
- OffPageConnectorFormat object
- TextRange object

See Also Font object

FullName Property

Syntax <*Object Name*>.**FullName**

Data Type String (read-only)

Description The FullName property gets the full path name of the specified object, including the file name. This property is used by several object types, which are listed in the following table. This property is read-only for all object types.

Object Name	Description of Caption Property
Application	For the Application object, the FullName property returns the full path to the iGrafx Professional executable file.
Diagram	For the Diagram object, the FullName property returns the full path to the iGrafx Professional (.igx) file in which the diagram is stored.
Document	For the Document object, the FullName property returns the full path to the iGrafx Professional (.igx) file.
ExtensionProject	For the ExtensionProject object, the FullName property returns the full path to the file that contains the specified extension project.
Template	For the Template object, the FullName property returns the full path to the specified template file.

See Also Caption property, Name property, Path property, Version property, Build property

Height Property

Topic Under Construction!!!

Syntax <*Object Name*>.**Height**

Data Type Integer, Long, or Double, depending on object type (see table below)

Description The Height property specifies the height of a particular object. The property is common to a number of iGrafx Professional API objects; however, its meaning and purpose vary slightly depending on the object. The following table lists the objects that have a Height property, and provides information about the data type and a description.

Object Name	Data Type	Description of Height Property
Application	Long Read/Write	For the Application object, the Height property specifies the height of the application window. The value is specified in pixels.
DiagramObject	Long Read/Write	For the DiagramObject object, the Height property specifies the height of the object. The value is specified in twips (1440 twips = 1 inch).
DiagramView	Long Read/Write	For the DiagramView object, the Height property specifies the height of the view
EllipseGraphic	Double Read/Write	For the EllipseGraphic object, the Height property specifies the height
ImageGraphic	Double Read-Only	For the ImageGraphic object, the Height property specifies
ObjectRange	Long Read-Only	For the ObjectRange object, the Height property returns the height, in twips (1440 twips = 1 inch), of the entire range of objects.
Page	Long Read-Only	For the Page object, the Height property returns the height of the page
PopupWindow	Integer Read/Write	For the PopupWindow object, the Height property specifies
RectangleGraphic	Double Read/Write	For the RectangleGraphic object, the Height property specifies
ShapeClass	Long Read/Write	For the ShapeClass object, the Height property specifies
Window	Long Read/Write	For the Window object, the Height property specifies

Related Width property, Left property, Top property

Error IGRAFX_E_MUSTBEPOSITIVE

Name Property

Topic Under Construction!!!

Syntax <*Object Name*>.**Name**

Data Type String (read/write or read-only, depending on object—see table)

Description The Name property specifies the name for a particular object. This property is used by a number of objects in the iGrafx Professional API. For some objects, the property is read/write, and for others it is read-only. The following table lists the objects that have a Name property, and describes the meaning and purpose of the property for each object.

Object Name	Access Type	Description of Height Property
Application	Read-Only	For the Application object, the Name property returns the name of the application. You can only get the application name; you cannot change it. See the Example section. Note that the Application object also has a FullName property.
CommandCategory	Read/Write	For the CommandCategory object, the Name property specifies the name of a particular category of commands. This could be a built-in category, or a custom category created by you or some other developer.
Component	Read/Write	For the CommandCategory object, the Name property specifies
CustomDataDefinition	Read/Write	For the CustomDataDefinition object, the Name property specifies
CustomDataValue	Read-Only	For the CustomDataValue object, the Name property specifies
DecisionCase	Read/Write	For the DecisionCase object, the Name property specifies
Diagram	Read/Write	For the Diagram object, the Name property specifies Note that the Diagram object also has a FullName property.
DiagramObject	Read/Write	For the DiagramObject object, the Name property specifies
Document	Read/Write	For the Document object, the Name property specifies Note that the Document object also has a FullName property.
Entity	Read/Write	For the Entity object, the Name property specifies the name of the entity. Entity names are useful for tracking an entity's progress through a diagram.
Font	Read/Write	For the Font object, the Name property specifies the name of a font.
Layer	Read/Write	For the Layer object, the Name

		property specifies the name of the layer. This name is displayed on a Tab at the bottom of the Document or Diagram window.
Path	Read-Only	For the Path object, the Name property specifies
Property	Read-Only	For the Property object, the Name property specifies
PropertyList	Read-Only	For the PropertyList object, the Name property specifies
ShapeClass	Read/Write	For the ShapeClass object, the Name property specifies
Template	Read-Only	For the Template object, the Name property specifies Note that the Template object also has a FullName property.

Related Caption property, Build property, Version property, Path property, FullName property

Parent Property

Syntax <*Object Name*>.**Parent**

Data Type Object of the type that is the parent of the current object (see Table)

Description The Parent property returns the parent object of an object. All objects in the iGrafx Professional API have a Parent property except the following:

- Change object
- CommandHandler object
- IGrafxExtension object

The Parent property's value can be one of many types of objects, depending on which object you call the Parent property from. Many objects return the same object as their parent. The Parent property always returns an object, and is therefore, read-only (see Object Properties for more information).

The following table lists the parent object type for all iGrafx Professional API objects. In many cases, objects that return the same parent object are grouped. Note that another way to determine the parent of an object is to look at the iGrafx API Object Hierarchy diagram, or use the Visual Basic editor's Object Browser.

Object Name	Object Returned by the Parent Property
Adjustment, AnyControls, Application, ChangeBracket, CommandBarControl, CommandBars, CommandCategories, CommandCategory, ConnectorLine, Cursor, Document, Documents, EventManager, FloatingWindows, Font, FontNames, GeometryHelper, GraphicBuilder, Grid, OutputWindow, PercentGauge, Point, Points, RecentFiles, Ruler, ShapeLibraries, ShapeLibraryItem, StatusBar, Template, Templates, Window, Windows, Workspace	Application object
CommandBarItem	CommandBar object
CommandBarCommand	CommandBarItem object
ConnectPoint	ConnectPoints object
CustomDataDefinition	CustomDataDefinitions object
CustomDataValue	CustomDataValues object
DecisionCase	DecisionCases object
DepartmentRange, DiagramObjects, DiagramView, Entity, Guideline, Guidelines, Layer, Layers, LinkIndicatorStyle, NoteIndicatorStyle, ObjectRange, OffPageConnectorFormat, OleObject, Shape, StartPointNames, TextGraphicObject	Diagram object
BlockFormat, ChildTextBlock, ChildTextBlocks, Fields, FieldText, TextBlock, TextRange	DiagramObject object
DiagramType	DiagramTypes object
Component, Components, CustomDataDefinitions, Diagram,	Document object

Diagrams, Entities	
Field	Fields object
GalleryPane	GalleryPanes object
EllipseGraphic, GraphicGroup, ImageGraphic, PolygonGraphic, PolygonPoint, PolyPolygonGraphic, RectangleGraphic	Graphic object
Graphics	GraphicGroup object
OutputPane	OutputPanes object
OutputPanes	OutputWindow object
Path	Paths object
PolygonPoints	PolygonGraphic object
Property	PropertyList object
Adjustments, CustomDataValues, DecisionCases, ExcludedDepartmentNames, Link, Links, Note, Paths	Shape object
ConnectPoints	ShapeClass object
FieldTexts	TextRange object

Path Property

Topic Under Construction!!!

Syntax <*Object Name*>.**Path**

Data Type String (read-only)

Description The Path property

The following objects have a Path property:

- Document object
- ExtensionProject object
- Template object

Text Property

Topic Under Construction!!!

Syntax <*Object Name*>.**Text**

Data Type String (read/write)

Description The Text property specifies the text string from a particular object. The property ignores carriage returns and line feeds. See TextLF if you need returns and LineFeeds. When you read the property, all carriage returns and line feeds are converted to spaces.

This property is used by the following objects:

- ChildTextBlock object
- Department object
- HeaderFooter object
- LinkIndicatorStyle object
- Note object
- NoteIndicatorStyle object
- Paragraph object
- PercentGauge object
- Shape object
- StatusBar object
- TextBlock object
- TextGraphicObject object
- TextRange object

TextLF Property

Topic Under Construction!!!

Syntax <*Object Name*>.**TextLF**

Data Type String (read/write)

Description The TextLF property gets or sets the text string from a particular object. The property, unlike the Text property, preserves any carriage returns and line feeds. See the Text property if you need to ignore carriage returns and line feeds.

This property is used by the following objects:

- ChildTextBlock object
- Department object
- HeaderFooter object
- Note object
- Paragraph object
- Shape object
- TextBlock object
- TextGraphicObject object
- TextRange object

Top Property

Topic Under Construction!!!

Syntax <*Object Name*>.**Top**

Data Type Long or Double, depending on object (read/write)

Description The Top property specifies the position of the top of an object. This property is used by a number of objects in the iGrafx Professional API. The property has essentially the same meaning for all objects, although the data type and the units vary. These differences are presented in the following table.

Object Name	Data Type	Description of Height Property
Application	Long Read/Write	For the Application object, the Top property specifies
ArcGraphic	Double Read/Write	For the ArcGraphic object, the Top property specifies
CommandBar	Integer Read/Write	For the CommandBar object, the Top property specifies
DiagramObject	Long Read/Write	For the DiagramObject object, the Top property specifies
DiagramView	Long Read/Write	For the DiagramView object, the Top property specifies
EllipseGraphic	Double Read/Write	For the EllipseGraphic object, the Top property specifies
Gallery	Integer Read/Write	For the Gallery object, the Top property specifies
ObjectRange	Long Read/Write	For the ObjectRange object, the Top property specifies
Page	Long Read-Only	For the Page object, the Top property specifies
RectangleGraphic	Double Read/Write	For the RectangleGraphic object, the Top property specifies
TextBlock	Double Read/Write	For the TextBlock object, the Top property specifies
Window	Long Read/Write	For the Window object, the Top property specifies

Related Width property, Height property, Left property, Window property, Move event

Visible Property

Topic Under Construction!!!

Syntax <*Object Name*>.**Visible**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Visible property specifies whether an object is visible in the interface. Setting this property to False hides the object, or causes it not to be displayed. Setting the property to True causes the object to be displayed.

The Visible property is common to a number of different objects in the iGrafx Professional API. Its purpose and meaning are the same for all objects, and the property is read/write for all objects. The following tables lists the objects that have a Visible property, and provides a description of the property's use for each object.

Object Name	Description of Visible Property
Application	For the Application object, the Visible property specifies
CommandBar	For the CommandBar object, the Visible property specifies
CommandBarItem	For the CommandBarItem object, the Visible property specifies
Gallery	For the Gallery object, the Visible property specifies
GalleryPane	For the GalleryPane object, the Visible property specifies
Grid	For the Grid object, the Visible property specifies
Guidelines	For the Guidelines object, the Visible property specifies
Layer	For the Layer object, the Visible property specifies
OutputWindow	For the OutputWindow object, the Visible property specifies
PercentGauge	For the PercentGauge object, the Visible property specifies
Ruler	For the Ruler object, the Visible property specifies
StatusBar	For the StatusBar object, the Visible property specifies
Window	For the Window object, the Visible property specifies

Width Property

Topic Under Construction!!!

Syntax <*Object Name*>.**Width**

Data Type Long or Double, depending on object (read/write)

Description The Width property specifies how wide to make a particular object. This property is common to a number of objects in the iGrafx Professional API. Its meaning and purpose are fundamentally the same for each object; however, its data type, accessibility, and units vary depending on the object to which it belongs.

The following table lists the objects that have a Width property, and provides descriptions for the meaning and use of the property for each object.

Object Name	Data Type	Description of Width Property
Application	Long Read/Write	For the Application object, the Width property specifies
DiagramObject	Long Read/Write	For the DiagramObject object, the Width property specifies
DiagramView	Long Read/Write	For the DiagramView object, the Width property specifies
EllipseGraphic	Double Read/Write	For the EllipseGraphic object, the Width property specifies
ImageGraphic	Double Read-Only	For the ImageGraphic object, the Width property specifies
LineFormat	Double Read/Write	For the LineFormat object, the Width property specifies
ObjectRange	Long Read-Only	For the ObjectRange object, the Width property specifies
Page	Long Read-Only	For the Page object, the Width property specifies
RectangleGraphic	Double Read/Write	For the RectangleGraphic object, the Width property specifies
ShapeClass	Long Read/Write	For the ShapeClass object, the Width property specifies
Window	Long Read/Write	For the Window object, the Width property specifies

Related Height property, Left property, Top property

Errors IGRAFX_E_MUSTBEPOSITIVE

Delete Method

Topic Under Construction!!!

Syntax <*Object Name*>.**Delete**

Description The Delete method deletes the specified object. This method is common to many objects in the iGrafx Professional API. Its meaning and purpose are the same for every object. In most cases, the Delete method removes an object from a collection.

Object Name	Description of Delete Method
Adjustment	For the Adjustment object, the Delete method removes an Adjustment object from the Adjustments collection, and therefore, from a Shape.
CommandBar	For the CommandBar object, the Delete method removes a CommandBar from the CommandBars collection.
CommandBarItem	For the CommandBarItem object, the Delete method removes the item from the CommandBarItems collection.
CommandCategory	For the CommandCategory object, the Delete method removes a command category from the CommandCategories collection.
Component	For the Component object, the Delete method removes a Component from the Components collection.
ConnectorLine	For the ConnectorLine object, the Delete method removes the specified ConnectorLine object from a diagram.
ConnectPoint	For the ConnectPoint object, the Delete method removes a connect point from the ConnectPoints collection.
CustomDataDefinition	For the CustomDataDefinition object, the Delete method removes a custom data definition from the CustomDataDefinitions collection.
DecisionCase	For the DecisionCase object, the Delete method removes a decision case from the DecisionCases collection.
Department	For the Department object, the Delete method removes a department from the diagram's Departments collection.
Entity	For the Field object, the Delete method removes an entity from the Entities collection.
Field	For the Field object, the Delete method removes a field from the Fields collection.
FieldText	For the FieldText object, the Delete method removes
Graphics	For the Graphics object, the Delete method removes
Guideline	For the Guideline object, the Delete method removes
Layer	For the Layer object, the Delete method removes
Link	For the Link object, the Delete method removes
ObjectRange	For the ObjectRange object, the Delete method removes
OutputPane	For the OutputPane object, the Delete method

	removes
Property	For the Property object, the Delete method removes
PropertyList	For the PropertyList object, the Delete method removes
ShapeLibraryItem	For the ShapeLibraryItem object, the Delete method removes

Graphic Object

The Graphic object is the object that represents the appearance and characteristics of the graphical portion of a Shape or TextGraphicObject object. The Graphic object is a property of the following objects (refer to the Object Hierarchy):

- Shape object
- ShapeClass object
- TextGraphicObject object
- GraphicBuilder object

The Graphic object has a number of formatting properties in common with both the Shape object and the TextGraphicObject object, such as fill and line formatting. The properties at Shape or TextGraphicObject level always overrides those at the Graphic level unless the ProtectFillFormat and ProtectLineFormat properties of the Graphic object are set to True.

For more information about the relationship between the Shape object and the Graphic object, refer to the discussion of the Shape object.

Creating and Using Graphic Objects

To create a new Graphic object, use the GraphicBuilder object. Once you have created a graphic with the appearance you want, you can make use of it in one of the following ways:

- Add it to a diagram as a TextGraphicObject using the DiagramObjects.AddGraphic method. In this case, any fill or line formats set for the Graphic object are preserved, even if the ProtectFillFormat and ProtectLineFormat properties are set to False.
- Add it to an existing TextGraphicObject that only has text using either of the following:

```
igxTextGraphic.Graphic.Replace igxGraphicBuilder.Graphic
OR
igxTextGraphic.Graphic = igxGraphicBuilder.Graphic
```

- Replace the graphical part of an existing TextGraphicObject using the Replace method or direct assignment (see the previous bullet item).
- Replace the graphical part of an existing Shape object using the Replace method or direct assignment.
- Replace the graphical part of an existing ShapeClass object using the Replace method or direct assignment. Refer to the discussion of the ShapeClass object for more information about the effects of doing this.
- Convert the graphic to a Shape object by first creating a TextGraphicObject with the graphic, and then using the TextGraphicObject.ConvertToShape method.
- Create a new ShapeLibraryItem (a Shape) with the ShapeLibrary.AddFromGraphic method.

When replacing an existing graphical part of either a Shape or TextGraphic object with a new graphic object, you must set the ProtectFillFormat and ProtectLineFormat properties of the new graphic object for its fill and line formats to be preserved. The fill and line formats of the existing shape or TextGraphicObject override the Graphic object's properties unless the "Protect" properties are set to True.

When you change a graphic into a shape with either the TextGraphicObject.ConvertToShape method or the ShapeLibrary.AddFromGraphic method, you can then create a ShapeClass object from your new shape by using the ShapeLibrary.Add method with the AddUnique argument set to True. For more information, refer to the ShapeLibrary and ShapeClass objects.

In addition, a shape can be converted to a graphic. However, be aware that a Graphic object does not have the same functionality of a Shape object. If you convert a shape that has VBA code or other "intelligence" associated with it, all of that functionality is lost and cannot be regained.

Grouping Graphic Objects

Graphic objects can be grouped; that is, two or more graphic primitives can be combined to create more elaborate graphical symbols to use as shapes (refer to the GraphicGroup object). In addition, polygon primitives can be joined in a collection called a PolyPolygonGraphic, which allows you to create shapes with cut-outs. For more

information about creating graphical shapes in iGrafx Professional, refer to the iGrafx Professional User's Guide.

Determining a Graphic Object's Type

The Type property informs you of the type of the Graphic object. A Graphic object can be a rectangle, an ellipse, a polygon, a polypolygon, an arc, an image (bitmap), or a metafile. For example, if the Type property is ixGraphicEllipse, then the EllipseGraphic object is accessible; other graphic primitive objects (RectangleGraphic, PolygonGraphic, etc.) are not accessible, and attempting to access any of their properties or methods returns an error (for instance, IGRAFX_E_NOTANELLIPSE). The programmer should always check the value of the Type property before attempting to access any properties of a graphic primitive object.

Properties, Methods, and Events

All of the properties, methods, and events for the Graphic object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	GetImage	
ArcGraphic	Replace	
EllipseGraphic	ResetCoordinateSpace	
FillFormat	SetCoordinateSpace	
GraphicGroup	SetImage	
ImageGraphic		
LineFormat		
MetafileGraphic		
Parent		
PolygonGraphic		
PolyPolygonGraphic		
ProtectFillFormat		
ProtectLineFormat		
RectangleGraphic		
Type		

ArcGraphic Property

Syntax *Graphic*.**ArcGraphic**

Data Type ArcGraphic object (read-only, See Object Properties)

Description The ArcGraphic property returns an ArcGraphic object if the Graphic.Type property equals ixGraphicArc. If the Type property is not ixGraphicArc, then attempting to access the ArcGraphic object results in an error.

Error Returns IGRAFX_E_NOTANARC if you try to access the ArcGraphic object and the Type property is not equal to ixGraphicArc.

Example The following example adds a shape to the diagram, and builds an ArcGraphic. Then shape's graphic is replaced with the ArcGraphic. The ArcGraphic is then modified while inside the shape.

```
' Dimension the variables
Dim igxBuilder As New GraphicBuilder
Dim igxShape As Shape
' Add a new shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Build an ArcGraphic
igxBuilder.Arc 0, 0, 1, 1, 0, 1, 1, 1, True
' Replace the shape's graphic with the ArcGraphic
igxShape.Graphic.Replace igxBuilder.Graphic
' Check the shape's graphic type
If igxShape.Graphic.Type = ixGraphicArc Then
   MsgBox "Click OK to raise the Bottom of the ArcGraphic."
   ' Move the Bottom property of the ArcGraphic
   igxShape.Graphic.ArcGraphic.Bottom = 0.5
End If
MsgBox "Click OK to continue."
```

See Also Type property

ArcGraphic object

GraphicBuilder object

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

EllipseGraphic Property

Syntax *Graphic*.**EllipseGraphic**

Data Type EllipseGraphic object (read-only, See Object Properties)

Description The EllipseGraphic property returns an EllipseGraphic object if the Graphic.Type property equals ixGraphicEllipse. If the Type property is not ixGraphicEllipse, then attempting to access the EllipseGraphic object results in an error.

Errors Returns IGRAFX_E_NOTANELLIPSE if you try to access the EllipseGraphic object and the Type property is not equal to ixGraphicEllipse.

Example The following example creates a shape in the active diagram and replaces the graphic of the shape with one rectangle and one ellipse that are created using the GraphicBuilder object. It then uses the RectangleGraphic property to change the rectangle's size and placement and the EllipseGraphic property to change the size of the ellipse.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim iCount As Integer
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic
igxGrfxBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Set the fill color of the rectangle to red
igxGrfxBuilder.Graphic.FillFormat.FillColor = vbRed
' Add an ellipse to the graphic
igxGrfxBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the fill color of the ellipse to blue
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Display message box before changing the ellipse and the rectangle
MsgBox "Click to shrink the ellipse and enlarge the rectangle."
' Go through the graphics in the shape
For iCount = 1 To igxShape.Graphic.GraphicGroup.Graphics.Count
    ' Get the graphic from the group
    Set igxGraphic = igxShape.Graphic.GraphicGroup. _
        Graphics.Item(iCount)
    ' Determine which shape to change
    Select Case igxGraphic.Type
        ' Handle the ellipse case
        Case ixGraphicEllipse
            igxGraphic.EllipseGraphic.Height = 0.25
            igxGraphic.EllipseGraphic.Width = 0.25
```

```
            ' Handle the rectangle case
            Case ixGraphicRectangle
                igxGraphic.RectangleGraphic.Height = 0.75
                igxGraphic.RectangleGraphic.Width = 0.75
                igxGraphic.RectangleGraphic.Left = 0.25
                igxGraphic.RectangleGraphic.Top = 0.25
        End Select
    Next iCount
```

See Also

Type property

EllipseGraphic object

GraphicBuilder object

TextGraphicObject object

Shape object

iGrafx API Object Hierarchy

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

FillFormat Property

Syntax *Graphic*.**FillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The FillFormat property returns the FillFormat object for the specified Graphic object. This object is used to define the fill formatting characteristics for a graphic. The FillFormat object controls whether a fill is used, and if so, what type of fill (solid, pattern, or gradient), and the color or colors used.

Example The following example creates a shape on the active diagram and then creates a GraphicBuilder object to replace the graphic of the shape. The graphic builder creates a rectangle and ellipse and then fills each with color using the fill format of each of the graphic objects.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxDiagramObj As DiagramObject
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic
igxGrfxBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Set the fill color of the rectangle to red
igxGrfxBuilder.Graphic.FillFormat.FillColor = vbRed
' Add an ellipse to the graphic
igxGrfxBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the fill color of the ellipse to blue
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

GetImage Method

Syntax *Graphic*.**GetImage**(*Width* As Long, *Height* As Long) As StdPicture

Description The GetImage method returns a stdPicture object that represents the graphic of the shape. The returned stdPicture object can be used in a Visual Basic image control, command bar button graphic, or even set as the image of a shape. For more information on the stdPicture object refer to the Visual Basic help.

The *Width* argument represents the width of the image to return from the graphic. The units of measure are pixels.

The *Height* argument represents the height of the image to return from the graphic. The units of measure are pixels.

Example The following example creates a shape in the active diagram, and then gets an image representation of the shape from its graphic object and stores it in the ShapeImage variable. It is important to note the conversion technique, using the DiagramView.PointToScreen method, that converts the twips units into pixels.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim igxShapeImage As StdPicture
Dim lTop As Long
Dim lBottom As Long
Dim lLeft As Long
Dim lRight As Long
Dim lScreenLeft As Long
Dim lScreenRight As Long
Dim lScreenTop As Long
Dim lScreenBottom As Long
Dim lWidth As Long
Dim lHeight As Long
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Set the fill color of the shape to red
igxShape.FillColor = vbRed
' Get the left, right, top, and bottom positions of the shape and
' convert those positions to pixels
lLeft = igxShape.DiagramObject.Left
lTop = igxShape.DiagramObject.Top
lRight = igxShape.DiagramObject.Right
lBottom = igxShape.DiagramObject.Bottom
' Convert the twips to pixels
ActiveDiagram.Views.Item(1).DiagramView.PointToScreen lLeft, _
    lTop, lScreenLeft, lScreenTop
ActiveDiagram.Views.Item(1).DiagramView.PointToScreen lRight, _
    lBottom, lScreenRight, lScreenBottom
' Calculate the Width and Height in pixels from
' the returned values
lWidth = lScreenRight - lScreenLeft
lHeight = lScreenBottom - lScreenTop
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Get the stdPicture object that represents the image of the shape
```

```
Set igxShapeImage = igxGraphic.GetImage(lWidth, lHeight)
MsgBox "View the diagram"
```

See Also

SetImage method

DiagramView.PointToScreen method

iGrafx API Object Hierarchy

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

GraphicGroup Property

Syntax *Graphic*.**GraphicGroup**

Data Type GraphicGroup object (read-only, See Object Properties)

Description The GraphicGroup property returns a GraphicGroup object if the Graphic.Type property equals ixGraphicGroup. If the Type property is not ixGraphicGroup, then attempting to access the GraphicGroup object results in an error.

Errors Returns IGRAFX_E_NOTAGROUP if you try to access the GraphicGroup object and the Type property is not equal to ixGraphicGroup.

Example The following example

See Also Type property

GraphicGroup object

iGrafx API Object Hierarchy

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

ImageGraphic Property

Syntax *Graphic*.**ImageGraphic**

Data Type ImageGraphic object (read-only, See Object Properties)

Description The ImageGraphic property returns an ImageGraphic object if the Graphic.Type property equals ixGraphicImage. If the Type property is not ixGraphicImage, then attempting to access the ImageGraphic object results in an error. The ImageGraphic object allows you to use a bitmap image as a graphic for either a Shape or a TextGraphicObject object.

Error Returns IGRAFX_E_NOTANIMAGE if you try to access the ImageGraphic object and the Type property is not equal to ixGraphicImage.

Example The following example sets a shape's graphic to an image loaded from disk. It then displays the ImageGraphic object's width and height, which is in pixels.

```
' Dimension the variables
Dim igxShape As Shape
' Add a new shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Make the shape larger
igxShape.DiagramObject.Width = 1440 * 4
igxShape.DiagramObject.Height = 1440 * 2.5
igxShape.DiagramObject.Left = 1440
igxShape.DiagramObject.Top = 1440
MsgBox "Click OK to load a bitmap."
' Load a bitmap from disk. The LoadPicture function returns an
' IPictureDisp, a StdPicture. LoadPicture belongs to the
' VBA StdOLE library
igxShape.Graphic.SetImage LoadPicture("c:\winnt\winnt256.bmp")
' Display the width and height of the image
MsgBox "The image is " & igxShape.Graphic.ImageGraphic _
   .Width & " X " & igxShape.Graphic.ImageGraphic.Height _
   & " pixels"
MsgBox "Click OK to continue."
```

See Also Type property

ImageGraphic object

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

LineFormat Property

Syntax *Graphic*.**LineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The LineFormat property returns the LineFormat object for the specified Graphic object. This object is used to define the line formatting characteristics for a graphic. The LineFormat object controls the line style (solid, dashed, dotted, etc.), width, and color.

Example The following example creates a shape on the active diagram and then replaces its graphic with the graphic in the GrapihcsBuilder object. It then changes the line of the rectangle graphic to be blue, dashed, and 2 points wide using its LineFormat object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim igxLineFmt As LineFormat
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape on the active diagram.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic
igxGrfxBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Set the fill color of the rectangle to red
igxGrfxBuilder.Graphic.FillFormat.FillColor = vbRed
' Add an ellipse to the graphic
igxGrfxBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the fill color of the ellipse to green
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbGreen
' Get the LineFormat object for the rectangle graphic
Set igxLineFmt = igxGrfxBuilder.Graphic.GraphicGroup. _
    Graphics.Item(1).LineFormat
' Set the line color to blue
igxLineFmt.Color = vbBlue
' Set the line style to dashed
igxLineFmt.Style = ixLineDashed
' Set the line width to 2
igxLineFmt.Width = 20
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the diagram"
```

See Also LineFormat object

iGrafx API Object Hierarchy

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

MetafileGraphic Property

Syntax *Graphic*.**MetafileGraphic**

Data Type MetafileGraphic object (read-only, See Object Properties)

Description The MetafileGraphic property returns a MetafileGraphic object if the Graphic.Type property equals ixGraphicMetafile. If the Type property is not ixGraphicMetafile, then attempting to access the MetafileGraphic object results in an error. The MetafileGraphic object allows you to use a vector image as a graphic for a Shape or TextGraphicObject object.

Error Returns IGRAFX_E_NOTAMETAFILE if you try to access the MetafileGraphic object and the Type property is not equal to ixGraphicMetafile.

Example The following example converts a MetafileGraphic to a GraphicGroup. It then uses the GraphicGroup count property to report the number of graphic elements in the graphic.

The example requires at least one Clipart image inserted into the diagram.

```
' Dimension the variables
Dim igxMetafileGraphic As MetafileGraphic
Dim igxShape As Shape
Dim igxGraphicGroup As GraphicGroup
' Find the first Shape with MetafileGraphic in the diagram
With ActiveDiagram.DiagramObjects
   For Index = 1 To .Count
      If (.Item(Index).Type = ixObjectShape) Then
         If .Item(Index).Shape.Graphic.Type = _
         ixGraphicMetafile Then
             ' If found, get the Shape Object
             Set igxShape = .Item(Index).Shape
             Exit For
         End If
      End If
   Next Index
End With
igxShape.Graphic.MetafileGraphic.ConvertToGroup
Set igxGraphicGroup = igxShape.Graphic.GraphicGroup
MsgBox "The converted group has " & igxGraphicGroup.Graphics.Count _
    & " graphic elements."
```

See Also Type property

MetafileGraphic object

iGrafx API Object Hierarchy

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

PolygonGraphic Property

Syntax *Graphic*.**PolygonGraphic**

Data Type PolygonGraphic object (read-only, See Object Properties)

Description The PolygonGraphic property returns a PolygonGraphic object if the Graphic.Type property equals ixGraphicPolygon. If the Type property is not ixGraphicPolygon, then attempting to access the PolygonGraphic object results in an error.

Error Returns IGRAFX_E_NOTAPOLYGON if you try to access the PolygonGraphic object and the Type property is not equal to ixGraphicPolygon.

Example The following example creates a polygon with the GraphicBuilder object. It then determines the type of polygon, and displays the result.

```
' Dimension the variables
Dim igxBuilder As New GraphicBuilder
Dim igxShape As Shape
' Add a new shape
MsgBox "Click OK to make a polygon."
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Build a PolygonGraphic
igxBuilder.Polygon 0.5, 0.5, 0.5, 6
' Replace the shape's graphic with the PolygonGraphic
igxShape.Graphic.Replace igxBuilder.Graphic
' Check the shape's graphic type
If igxShape.Graphic.Type = ixGraphicPolygon Then
    ' Figure out what type of polygon
    Dim sString As String
    ' Set default text
    sString = "an unnamed polygon"
    ' Check the number of points in the polygon
    Select Case igxShape.Graphic.PolygonGraphic.PolygonPoints.Count
        Case 3:
            sString = "a triangle"
        Case 4:
            sString = "a rectangle"
        Case 5:
            sString = "a pentagon"
        Case 6:
            sString = "a hexagon"
        Case 8:
            sString = "an octagon"
     End Select
End If
MsgBox "The PolygonGraphic is " & sString & "."
```

See Also

Type property

PolygonGraphic object

GraphicBuilder object

iGrafx API Object Hierarchy

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

PolyPolygonGraphic Property

Syntax *Graphic*.**PolyPolygonGraphic**

Data Type PolyPolygonGraphic object (read-only, See Object Properties)

Description The PolyPolygonGraphic property returns a PolyPolygonGraphic object if the Graphic.Type property equals ixGraphicPolyPolygon. If the Type property is not ixGraphicPolyPolygon, then attempting to access the PolyPolygonGraphic object results in an error. This object is used primarily to create graphics with holes or cutouts.

Error Returns IGRAFX_E_NOTAPOLYPOLYGON if you try to access the PolyPolygonGraphic object and the Type property is not equal to ixGraphicPolyPolygon.

Example The following example builds two polygons with the GraphicBuilder. A shape has it's graphic replaced with the new PolyPolygonGraphic. The graphic is then altered once inside the shape.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim igxPolyPolygon As PolyPolygonGraphic
' Declare the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create first polygon
igxGrfxBuilder.BeginPath
' Build the polygon
MsgBox "Click OK to build the PolyPolygon."
igxGrfxBuilder.MoveTo 0.4, 1
igxGrfxBuilder.LineTo 0.4, 0.3
igxGrfxBuilder.LineTo 0.6, 0.3
igxGrfxBuilder.LineTo 0.6, 1
igxGrfxBuilder.LineTo 0.4, 1
' Close the polygon so it can be filled
igxGrfxBuilder.Close
' Build another polygon
igxGrfxBuilder.MoveTo 0.4, 0.2
igxGrfxBuilder.LineTo 0.4, 0
igxGrfxBuilder.LineTo 0.6, 0
igxGrfxBuilder.LineTo 0.6, 0.2
igxGrfxBuilder.LineTo 0.4, 0.2
' Close the polygon so it can be filled
igxGrfxBuilder.Close
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Get the PolyPolygon object
Set igxPolyPolygon = igxShape.Graphic.PolyPolygonGraphic
```

```
' Move the polypolygon point to widen base of graphic
igxPolyPolygon.Item(1).PolygonPoints.Item(1).X = 0.3
' Move the polypolygon point to widen base of graphic
igxPolyPolygon.Item(1).PolygonPoints.Item(4).X = 0.7
MsgBox "Click OK to continue."
```

See Also

Type property

PolyPolygonGraphic object

GraphicBuilder object

iGrafx API Object Hierarchy

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

ProtectFillFormat Property

Syntax *Graphic*.**ProtectFillFormat**[= {**True | False**}]

Data Type Boolean (read/write)

Description The ProtectFillFormat property specifies whether the fill format of the Graphic object is protected from changes specified through other API objects, or by a user. This property provides the same functionality as the checkbox in the Edit Symbol dialog, accessed through iGrafx Share or through the Shape Library dialog. Setting this property to True is like putting a lock on the fill of the graphic. A value of False means the fill can be altered by other API objects or through the user interface.

This property is especially useful for graphics that are constructed from several graphic primitives, such as a square with a star inside it (a GraphicGroup).

Note that the “protect fill” lock can be turned off at any time programmatically (by setting this property to False) or through the user interface by using the Edit Symbol dialog.

Example The following example creates a shape on the active diagram and the replaces the graphic of the shape with the graphic created in the GraphicBuilder object. The rectangle on the new graphic has the ProtectFill and ProtectLine property turned on. The ProtectFill and ProtectLine property are turned off for the ellipse. To test how this property works, run the code below to create the object, then change the line style or fill of the shape. Notice how the fill and line style for the ellipse change, but they do not change for the rectangle.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic
igxGrfxBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Add an ellipse to the graphic
igxGrfxBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the fill color of the rectangle to red
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(1). _
    FillFormat.FillColor = vbRed
' Turn on protection for the fill of the rectangle
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(1). _
    ProtectFillFormat = True
' Turn on protection for the line of the rectangle
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(1). _
    ProtectLineFormat = True
' Set the fill color of the ellipse to blue
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Turn off protection for the fill of the ellipse
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    ProtectFillFormat = False
' Turn off protection for the line of the ellipse
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    ProtectLineFormat = False
' Replace the graphic inside the shape with the new graphic
```

```
igxGraphic.Replace igxGrfxBuilder.Graphic
CR = Chr(13) & Chr(13)
MsgBox "Both graphics are in one Shape." & CR & _
   "The Rectangle has ProtectFillFormat = True, the Circle " _
   & "does not." & CR & "Click OK to change the Shape fill " _
   & "and line formats."
igxShape.FillColor = vbBlue
igxShape.LineWidth = 40
MsgBox "Only the Circle changed. The Rectangle did not change" & _
   Chr(13) & "because its fill and line formats were protected."
```

See Also ProtectLineFormat property

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

ProtectLineFormat Property

Syntax *Graphic*.**ProtectLineFormat**[= {**True | False**}]

Data Type Boolean (read/write)

Description The ProtectLineFormat property specifies whether the line format of the Graphic object is protected from changes specified through other API objects, or by a user. This property provides the same functionality as the checkbox in the Edit Symbol dialog, accessed through iGrafx Share or through the Shape Library dialog. Setting this property to True is like putting a lock on the line formatting of the graphic. A value of False means the line can be altered by other API objects or through the user interface.

This property is especially useful for graphics that are constructed from several graphic primitives, such as a square with a star inside it (a GraphicGroup).

Note that the “protect line” lock can be turned off at any time programmatically (by setting this property to False) or through the user interface by using the Edit Shape—Graphic tab dialog.

Example The following example creates a shape on the active diagram and the replaces the graphic of the shape with the graphic created in the GraphicBuilder object. The rectangle on the new graphic has the ProtectFill and ProtectLine property turned on. The ProtectFill and ProtectLine property are turned off for the ellipse. To test how this property works, run the code below to create the object, then change the line style or fill of the shape. Notice how the fill and line style for the ellipse change, but they do not change for the rectangle.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic
igxGrfxBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Add an ellipse to the graphic
igxGrfxBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the fill color of the rectangle to red
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(1). _
    FillFormat.FillColor = vbRed
' Turn on protection for the fill of the rectangle
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(1). _
    ProtectFillFormat = True
' Turn on protection for the line of the rectangle
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(1). _
    ProtectLineFormat = True
' Set the fill color of the ellipse to blue
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Turn off protection for the fill of the ellipse
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    ProtectFillFormat = False
' Turn off protection for the line of the ellipse
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    ProtectLineFormat = False
' Replace the graphic inside the shape with the new graphic
```

```
igxGraphic.Replace igxGrfxBuilder.Graphic
CR = Chr(13) & Chr(13)
MsgBox "Both graphics are in one Shape." & CR & _
    "The Rectangle has ProtectLineFormat = True, the Circle " _
    & "does not." & CR & "Click OK to change the Shape fill " _
    & "and line formats."
igxShape.FillColor = vbBlue
igxShape.LineWidth = 40
igxShape.LineColor = vbGreen
MsgBox "Only the Circle changed. The Rectangle did not change" & _
   Chr(13) & "because its fill and line formats were protected."
```

See Also ProtectFillFormat property

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

RectangleGraphic Property

Syntax *Graphic*.**RectangleGraphic**

Data Type RectangleGraphic object (read-only, See Object Properties)

Description The RectangleGraphic property returns a RectangleGraphic object if the Graphic.Type property equals ixGraphicRectangle. If the Type property is not ixGraphicRectangle, then attempting to access the RectangleGraphic object results in an error.

Error Returns IGRAFX_E_NOTARECTANGLE if you try to access the RectangleGraphic object and the Type property is not equal to ixGraphicRectangle.

Example The following example creates a shape on the active diagram and replaces the graphic of the shape with one rectangle and one ellipse that is created using a GraphicsBuilder object. It then uses the RectangleGraphic property to change the rectangle's size and placement and the EllipseGraphic property to change the size of the ellipse.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim iCount As Integer
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic
igxGrfxBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Set the fill color of the rectangle to red
igxGrfxBuilder.Graphic.FillFormat.FillColor = vbRed
' Add an ellipse to the graphic
igxGrfxBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the fill color of the ellipse to blue
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the diagram"
' Display message box before changing the ellipse and the rectangle
MsgBox "Click to shrink the ellipse and enlarge the rectangle."
' Go through the graphics in the shape
For iCount = 1 To igxShape.Graphic.GraphicGroup.Graphics.Count
    ' Get the graphic from the group
    Set igxGraphic = igxShape.Graphic.GraphicGroup. _
        Graphics.Item(iCount)
    ' Determine which shape to change
    Select Case igxGraphic.Type
        ' Handle the ellipse case
        Case ixGraphicEllipse
            igxGraphic.EllipseGraphic.Height = 0.25
            igxGraphic.EllipseGraphic.Width = 0.25
            MsgBox "View the diagram"
        ' Handle the rectangle case.
        Case ixGraphicRectangle
```

```
                igxGraphic.RectangleGraphic.Height = 0.75
                igxGraphic.RectangleGraphic.Width = 0.75
                igxGraphic.RectangleGraphic.Left = 0.25
                igxGraphic.RectangleGraphic.Top = 0.25
                MsgBox "View the diagram"
        End Select
    Next iCount
    MsgBox "View the diagram"
```

See Also

Type property

RectangleGraphic object

GraphicBuilder object

iGrafx API Object Hierarchy

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

Replace Method

Syntax *Graphic*.**Replace**(*newVal* As Graphic)

Description The Replace method replaces the current graphic of a Shape object, ShapeClass object, or TextGraphicObject object with the graphic specified in the *newVal* argument. For example, if you place a decision shape in a diagram, but decide that you want that particular decision shape to be a diamond with a blue circle in the center, you would create the new graphic, and use the Replace method (Shape.Graphic.Replace).

Example The following example creates a simple decision diagram. Rather than a plain decision shape, you want a circle at the center of the diamond. The code illustrates how to use the Replace method to change the graphical part of a shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxSource As Shape
Dim igxDest As Shape
Dim igxConnLine As ConnectorLine
Dim igxTextGraphic As TextGraphicObject
Dim igxGraphic As Graphic
Dim igxDecisionCase As DecisionCase
Dim iCount As Integer
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape.Text = "Read Meter"
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440, Application.ShapeLibraries.Item(1).Item(3))
igxGrfxBuilder.BeginPath
igxGrfxBuilder.MoveTo 0.5, 0
igxGrfxBuilder.LineTo 0, 0.5
igxGrfxBuilder.MoveTo 0, 0.5
igxGrfxBuilder.LineTo 0.5, 1
igxGrfxBuilder.MoveTo 0.5, 1
igxGrfxBuilder.LineTo 1, 0.5
igxGrfxBuilder.MoveTo 1, 0.5
igxGrfxBuilder.LineTo 0.5, 0
igxGrfxBuilder.Close
igxGrfxBuilder.EndPath
' Create an ellipse
igxGrfxBuilder.Ellipse 0.2, 0.2, 0.6, 0.6
' Replace the shape's graphic
igxShape.Graphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the diagram"
' Set the shape’s text
igxShape.TextLF = "Amount" & Chr$(13) & "> 1500"
' Set the shape’s height and width
igxShape.DiagramObject.Height = 1440
igxShape.DiagramObject.Width = 1440 + 360
' Add two decision cases to the shape
Call igxShape.DecisionCases.Add("Yes")
Call igxShape.DecisionCases.Add("No")
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
```

```
    (1440 * 3, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
igxShape.Text = "Apply Usage Discount"
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 6, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape.Text = "Apply Standard Rate"
MsgBox "View the diagram"
'Connect the shapes
For iCount = 1 To 2
    If (iCount = 1) Then
        Set igxSource = ActiveDiagram.DiagramObjects(iCount).Shape
        Set igxDest = ActiveDiagram.DiagramObjects(iCount + 1).Shape
        Set igxConnLine = ActiveDiagram.DiagramObjects. _
            AddConnectorLine(ixRouteRightAngle, _
            ixRouteFlagFindEdge, igxSource, ixDirEast, _
            ixConnectRelativeToShape, , , igxDest, ixDirWest, _
            ixConnectRelativeToShape)
    ElseIf (iCount = 2) Then
        Set igxSource = ActiveDiagram.DiagramObjects(iCount).Shape
        Set igxDest = ActiveDiagram.DiagramObjects(iCount + 1).Shape
        Set igxConnLine = ActiveDiagram.DiagramObjects. _
            AddConnectorLine(ixRouteRightAngle, _
            ixRouteFlagFindEdge, igxSource, ixDirSouth, _
            ixConnectRelativeToShape, , , igxDest, ixDirNorth, _
            ixConnectRelativeToShape)
        Set igxSource = ActiveDiagram.DiagramObjects(iCount).Shape
        Set igxDest = ActiveDiagram.DiagramObjects(iCount + 2).Shape
        Set igxConnLine = ActiveDiagram.DiagramObjects. _
            AddConnectorLine(ixRouteRightAngle, _
            ixRouteFlagFindEdge, igxSource, ixDirEast, _
            ixConnectRelativeToShape, , , igxDest, ixDirWest, _
            ixConnectRelativeToShape)
    End If
Next iCount
MsgBox "View the diagram"
```

ResetCoordinateSpace Method

Syntax *Graphic*.**ResetCoordinateSpace**

Description The ResetCoordinateSpace method resets the coordinate space to encompass only the visible graphic objects. The size of the bounding box does not change, but the visible graphic inside is resized to fit within the bounding box. This method doess not affect a shape with a graphic type of ixGraphicImage.

Example The following example creates a shape on the active diagram, and then replaces the graphic of the shape with the graphic created with the GraphicBuilder object. It then displays a message box before resetting the coordinate space to encompass only the visible graphic objects.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim igxGrfxBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic: Left and Top set at 30% of
' the coordinate space, and 50% of coordinate space tall and wide
igxGrfxBuilder.Rectangle 0.3, 0.3, 0.5, 0.5
' Add an ellipse to the graphic: Left and Top set at 20% of the
' coordinate space, and 50% of coordinate space tall and wide
igxGrfxBuilder.Ellipse 0.2, 0.2, 0.5, 0.5
' Set the fill color of the rectangle to red
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(1). _
    FillFormat.FillColor = vbRed
' Set the fill color of the ellipse to blue
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Replace the shape's graphic with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the diagram"
' Select the Shape to display the bounding rectangle
igxShape.DiagramObject.Selected = True
MsgBox "View the diagram"
' Display a message box to reset coordinate space
MsgBox "Click OK to reset the coordinate space of the graphic."
Call igxShape.Graphic.SetCoordinateSpace(0, 0, 0.5, 0.5)
MsgBox "View the diagram"
igxShape.Graphic.ResetCoordinateSpace
MsgBox "View the diagram"
```

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

SetCoordinateSpace Method

Syntax *Graphic*.**SetCoordinateSpace** *X1* As Double, *Y1* As Double, *X2* As Double, *Y2* As Double

Description The SetCoordinateSpace method is used to expand or contract the coordinate space boundaries of a Graphic object. The arguments specify the boundaries of the new coordinate space for the graphic. Even though the coordinate space of the shape is expanded or contracted, the bounding box for the shape remains the same. This method does not affect a shape with a graphic type of ixGraphicImage.

Example The following example creates a shape, and then replaces its graphic with the graphic in the GraphicBuilder object. It then uses the SetCoordinateSpace method to expand the coordinate space of the graphic after the message box is displayed.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim igxGrfxBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic: Left and Top set at 30% of
' the coordinate space, and 50% of coordinate space tall and wide
igxGrfxBuilder.Rectangle 0.3, 0.3, 0.5, 0.5
' Add an ellipse to the graphic: Left and Top set at 20% of the
' coordinate space, and 50% of coordinate space tall and wide
igxGrfxBuilder.Ellipse 0.2, 0.2, 0.5, 0.5
' Set the fill color of the rectangle to red
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(1). _
    FillFormat.FillColor = vbRed
' Set the fill color of the ellipse to blue
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Replace the shape's graphic with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the diagram"
' Select the Shape to display the bounding rectangle
igxShape.DiagramObject.Selected = True
MsgBox "View the diagram"
' Display a message box to reset coordinate space
MsgBox "Click OK to reset the coordinate space of the graphic."
Call igxShape.Graphic.SetCoordinateSpace(0, 0, 0.5, 0.5)
MsgBox "View the diagram"
igxShape.Graphic.ResetCoordinateSpace
MsgBox "View the diagram"
' Display a message box.
MsgBox "Click OK to expand the coordinate space of the graphic."
' Set the coordinate space to be larger.
igxShape.Graphic.SetCoordinateSpace -0.5, -0.5, 1.5, 1.5
```

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

SetImage Method

Syntax *Graphic*.**SetImage** *newVal* As StdPicture

Description The SetImage method replaces the graphic of a shape with a bitmap image. Once a shape's graphic has been set to an image, the ResetCoordinateSpace and SetCoordinateSpace methods have no effect. This method should be used sparingly however, because the size of the shape and file become quite large when bitmaps are used for the graphic.

The *newVal* argument specifies the name of the bitmap file to use as the graphic. You must use the LoadPicture standard function (or some other function that returns a StdPicture object, or an IPictureDisp).

Example The following example creates a shape on the active diagram and then replaces the graphic of the shape with a bitmap image.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
igxShape.DiagramObject.Height = 1440 * 3
igxShape.DiagramObject.Width = 1440 * 3
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Replace the graphic in the shape with a bitmap image
igxGraphic.SetImage LoadPicture("E:\notmyjob.jpg")
MsgBox "View the diagram"
```

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

Type Property

Syntax *Graphic*.**Type**

Data Type IxGraphicType enumerated constant (read-only)

Description The Type property returns the type of the current graphic. This property is read-only, and is used to determine the graphic primitive sub-type of the specified Graphic object.

The IxGraphicType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixGraphicEllipse
1	ixGraphicPolygon
2	ixGraphicPolyPolygon
3	ixGraphicRectangle
4	ixGraphicImage
5	ixGraphicGroup
6	ixGraphicMetafile
7	ixGraphicArc

Example The following example creates a shape on the active diagram and then replaces the graphic of the shape with the graphic of the GraphicsBuilder object. It then goes through the graphic and displays a message in the Immediate window depending on the type of graphic.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim iCount As Integer
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic
igxGrfxBuilder.Rectangle 0.5, 0.5, 0.5, 0.5
' Set the fill color of the rectangle to red
igxGrfxBuilder.Graphic.FillFormat.FillColor = vbRed
' Add an ellipse to the graphic
igxGrfxBuilder.Ellipse 0, 0, 0.5, 0.5
' Set the fill color of the ellipse to blue
igxGrfxBuilder.Graphic.GraphicGroup.Graphics.Item(2). _
    FillFormat.FillColor = vbBlue
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Go through the graphics in the shape
For iCount = 1 To igxShape.Graphic.GraphicGroup.Graphics.Count
    ' Get the graphic from the group
```

```
    Set igxGraphic = igxShape.Graphic.GraphicGroup. _
        Graphics.Item(iCount)
    ' Determine which type is the graphic
    Select Case igxGraphic.Type
        ' Handle the ellipse case
        Case ixGraphicEllipse
            MsgBox "Graphic is an ellipse."
        ' Handle the rectangle case
        Case ixGraphicRectangle
            MsgBox "Graphic is a rectangle."
    End Select
Next iCount
```

See Also

ArcGraphic object

EllipseGraphic object

GraphicGroup object

ImageGraphic object

MetafileGraphic object

PolygonGraphic object

PolyPolygonGraphic object

RectangleGraphic object

{button Graphic object,JI(`igrafxrf.HLP',`Graphic_Object')}

Graphics Object

The Graphics object is a collection of individual Graphic objects. A Graphics collection is associated with the Shape object and the GraphicsBuilder object, and is accessible from the GraphicsGroup object.

The Graphics collection provides the following functionality for working with Graphic objects.

- The ability to access any Graphic object that has been added to the Graphics collection.
- The ability to determine how many Graphic objects are currently in the collection.
- The ability to delete a Graphic object from the Graphics collection.
- The ability to add a new Graphic object to the Graphics collection of the type designated by the method used.
- The ability to add a GraphicGroup (a collection of graphic objects) to the collection.
- The ability change the drawing order of graphic items in the collection, by moving an item to the front or to the back.

The Graphics object allows the programmer to get at individual graphic objects within a group. With this, a programmer can access the properties and methods of a specific graphic within a group, or alternately, add one or more new graphic objects to a group of graphics.

The Graphics object is subordinate to the Group object (see Object Hierarchy). Therefore, it only works with a graphic whose Type equals Group (ixGraphicGroup). As an example, a shape might have a graphic whose type is a group, which consists of several polygons. To access a specific polygon within the group, you must go through the Graphics object.

The following code shows this concept.

```
If Shape1.Graphic.Type = ixGraphicGroup Then
   If Shape1.Graphic.Group.Graphics.Item(1).Graphic.Type = ixGraphicPolygon
Then
      Msgbox("I'm a polygon within a group")
   End If
End If
```

Properties, Methods, and Events

All of the properties, methods, and events for the Graphics object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	AddGroup	
Parent	AddPicture	
	Delete	
	Item	
	MoveToBack	
	MoveToFront	

Related Topics

Graphic object

iGrafx API Object Hierarchy

Add Method

Topic Under Construction!!!

Syntax *Graphics*.**Add**(*Graphic* As Graphic, [*Left* As Double], [*Top* As Double], [*Width* As Double = 1], [*Height* As Double = 1])

Description The Add method adds a graphic object to the GraphicGroup object associated with the specified Graphics object. The method's arguments allow you to specify the type of graphic to add to the group, and set the position and size of the graphic.

The *Graphic* argument specifies the Graphic object that is to be added to the Graphics collection. A graphic can be created using the GraphicBuilder object.

The *Left* argument is used to set the position of the left side of the graphic. This value can range from 0.0 to 1.0, and is relative to the shape coordinate space, which is typically 0,0 to 1,1.

The *Top* argument is used to set the position of the top of the graphic. This value can range from 0.0 to 1.0, and is relative to the shape coordinate space, which is typically 0,0 to 1,1.

The *Width* argument is used to set the width of the graphic. This value can range from 0.0 to 1.0, which is relative to the shape coordinate space, which is typically 0,0 to 1,1.

The *Height* argument is used to set the height of the graphic. This value can range from 0.0 to 1.0, which is relative to the shape coordinate space, which is typically 0,0 to 1,1.

Example The following example

Still not a working example

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxGraphic As Graphic
Dim igxBuilder As New GraphicBuilder
' Create 2 new shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440, Application.ShapeLibraries(1)(3))
' Label the shapes
igxShape1.Text = "Shape 1"
igxShape2.Text = "Shape 2"
' Add 2 polygons to the GraphicBuilder
igxBuilder.Polygon 0.15, 0.5, 0.15, 4
igxBuilder.Polygon 0.35, 0.5, 0.25, 4, 90
' Set the igxGraphics variable to the GraphicBuilder graphics
Set igxGraphic = igxBuilder.Graphic
' Add the graphics from the two shapes to the Graphics collection
' of the GraphicGroup
igxGraphic.GraphicGroup.Graphics.Add _
   igxShape1.Graphic, 0.5, 0.7, 0.5, 0.3
igxGraphic.GraphicGroup.Graphics.Add _
   igxShape2.Graphic, 0.6, 0.3, 0.4, 0.3
MsgBox "The Graphics collection contains " _
    & igxGraphicGroup.Graphics.Count & " items."
' Replace the graphic in each shape with a graphic
' from the Graphics collection
```

```
igxShape1.Graphic.Replace igxGraphic.GraphicGroup.Graphics.Item(1)
MsgBox "Replaced shape graphic with first graphic in " _
    & "the Graphics collection."
igxShape1.Graphic.Replace igxGraphic.GraphicGroup.Graphics.Item(2)
MsgBox "Replaced shape graphic with second graphic in " _
    & "the Graphics collection."
' Pause for the user
MsgBox "Now replace shape 2's graphic with the entire GraphicGroup"
igxShape2.Graphic.Replace igxGraphic
```

See Also

GraphicBuilder object

iGrafx API Object Hierarchy

{button Graphics object,JI(`igrafxrf.HLP',`Graphics_Object')}

AddGroup Method

Topic Under Construction!!!

Syntax *Graphics*.**AddGroup**

This method is most likely not implemented. Build 76, 3/16/1999 ???

Description The AddGroup method creates an empty graphics group, which can then be filled with one or more primitive graphic types. Items of a group act in the same manner as objects that have been grouped using the tools in the iGrafx Professional interface.

Example The following example illustrates how to use the AddGroup method to create a new graphics group object, and populate it with several graphic primitives.

{button Graphics object,JI(`igrafxrf.HLP',`Graphics_Object')}

Item Method

Syntax *Graphics*.**Item**(*Index* As Integer) As Graphic

Description The Item method returns the Graphic object at the specified *Index* from the Graphics collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Graphic. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example creates four polygons with the GraphicBuilder, and assigns each one to a shape in the active diagram. Then the number of points is retreived for each polygon, using the Item method.

```
Private Sub Main()
    ' Dimension subroutine variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxShape3 As Shape
    Dim igxShape4 As Shape
    Dim igxGraphics As Graphics
    Dim igxBuilder1 As New GraphicBuilder
    ' Create 4 new shapes
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
    Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 5, 1440)
    Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 7, 1440)
    ' Label the shapes
    igxShape1.Text = "Triangle "
    igxShape2.Text = "Rectangle "
    igxShape3.Text = "Pentagon "
    igxShape4.Text = "Hexagon "
    ' Add 4 polygons to the GraphicBuilder
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 3, -90
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 4, 45
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 5, -90
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 6
    ' Set the igxGraphics variable to the GraphicBuilder1
    ' Graphics property
    Set igxGraphics = igxBuilder1.Graphic.GraphicGroup.Graphics
    ' Replace the graphic in each shape with a polygon
    ' from the igxGraphics object
    igxShape1.Graphic.Replace igxGraphics.Item(1)
    igxShape2.Graphic.Replace igxGraphics.Item(2)
    igxShape3.Graphic.Replace igxGraphics.Item(3)
    igxShape4.Graphic.Replace igxGraphics.Item(4)
    ' Retreive the point count for each polygon in igxGraphics
    ' and add it to the text on each shape
    igxShape1.Text = igxShape1.Text & igxGraphics.Item(1) _
        .PolygonGraphic.PolygonPoints.Count & " sides"
    igxShape2.Text = igxShape2.Text & igxGraphics.Item(2) _
        .PolygonGraphic.PolygonPoints.Count & " sides"
```

```
    igxShape3.Text = igxShape3.Text & igxGraphics.Item(3) _
        .PolygonGraphic.PolygonPoints.Count & " sides"
    igxShape4.Text = igxShape4.Text & igxGraphics.Item(4) _
        .PolygonGraphic.PolygonPoints.Count & " sides"
    ' Pause for the user
    MsgBox "Click OK to continue."
End Sub
```

{button Graphics object,JI(`igrafxrf.HLP',`Graphics_Object')}

MoveToBack Method

Syntax *Graphics*.**MoveToBack**(*Index* As Integer)

Description The MoveToBack method repositions the specified Graphic to the back in terms of the drawing order. An item at the back is drawn first, so other graphic objects may overlap or cover it. The *Index* argument specifies which graphic to move. Use the Count method to determine valid values for the *Index* argument.

The MoveToBack method has two implications:

- The specified Graphic object becomes the first member of the collection.
- The specified graphic is drawn first, meaning that other graphics in the collection overlay it.

When you move a graphic with this method, all the objects in the collection get new index numbers, reflecting the new order.

Example The following example creates a shape, and adds a rectangle and an ellipse to the shape using a GraphicBuilder. Initially, the rectangle is drawn first, so it is overlaid by the ellipse. The MoveToBack method is used to move the ellipse to the beginning of the Graphics collection so that the rectangle now overlays the ellipse.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBuilder1 As New GraphicBuilder
Dim igxGraphics As Graphics
' Add a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2.5, 1440 * 2.5)
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
' Add a Rectangle and Ellipse to the GraphicBuilder
igxBuilder1.Rectangle 0.25, 0.25, 0.75, 0.75
igxBuilder1.Ellipse 0, 0, 0.75, 0.75
' Replace the shape's graphic
igxShape.Graphic.Replace igxBuilder1.Graphic
Set igxGraphics = igxShape.Graphic.GraphicGroup.Graphics
' Move the Rectangle to the front
MsgBox "Click OK to make the Rectangle draw on top of the ellipse"
igxGraphics.MoveToBack 2
MsgBox "Click OK to continue"
```

The following example creates four shapes, and assigns polygon names to each. Then four polygons are created with the GraphicBuilder, and a Graphics object is set to it. The MoveToBack method is used to move the last polygon to the back, which alters the order of all the polygons in the collection. The polygons on the shapes, and the shapes' labels show the result.

```
Private Sub Main()
    ' Dimension subroutine variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxShape3 As Shape
   Dim igxShape4 As Shape
   Dim igxGraphics As Graphics
   Dim igxBuilder1 As New GraphicBuilder
```

```
    ' Create 4 new shapes
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 3, 1440)
    Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 5, 1440)
    Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
        (1440 * 7, 1440)
    ' Label the shapes
    igxShape1.Text = "Triangle "
    igxShape2.Text = "Rectangle "
    igxShape3.Text = "Pentagon "
    igxShape4.Text = "Hexagon "
    ' Add 4 polygons to the GraphicBuilder
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 3, -90
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 4, 45
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 5, -90
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 6
    ' Set the igxGraphics variable to the GraphicBuilder1
    ' Graphics property
    Set igxGraphics = igxBuilder1.Graphic.GraphicGroup.Graphics
    ' Move the last item to the back (making it the first item)
    igxGraphics.MoveToBack(4)
    ' Replace the graphic in each shape with a polygon
    ' from the igxGraphics object
    igxShape1.Graphic.Replace igxGraphics.Item(1)
    igxShape2.Graphic.Replace igxGraphics.Item(2)
    igxShape3.Graphic.Replace igxGraphics.Item(3)
    igxShape4.Graphic.Replace igxGraphics.Item(4)
    ' Pause for the user
    MsgBox "Notice the shift in the polygon order."
End Sub
```

{button Graphics object,JI(`igrafxrf.HLP',`Graphics_Object')}

MoveToFront Method

Syntax *Graphics*.**MoveToFront**(*Index* As Integer)

Description The MoveToFront method repositions the specified Graphic to the front in terms of the drawing order. An item at the front is drawn last, so it overlays and covers other graphic objects in the collection. The *Index* argument specifies which graphic to move. Use the Count method to determine valid values for the *Index* argument.

The MoveToFront method has two implications:

- The specified Graphic object becomes the last member of the collection.
- The specified graphic is drawn last, meaning that it overlays other graphics in the collection.

When you move a graphic with this method, all the objects in the collection get new index numbers, reflecting the new order.

Example The following example creates a shape, and adds a rectangle and an ellipse to the shape using a GraphicBuilder. Initially, the rectangle is drawn first, so it is overlaid by the ellipse. The MoveToFront method is used to move the ellipse to the beginning of the Graphics collection so that the rectangle now overlays the ellipse.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBuilder1 As New GraphicBuilder
Dim igxGraphics As Graphics
' Add a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2.5, 1440 * 2.5)
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
' Add a Rectangle and Ellipse to the GraphicBuilder
igxBuilder1.Rectangle 0.25, 0.25, 0.75, 0.75
igxBuilder1.Ellipse 0, 0, 0.75, 0.75
' Replace the shape's graphic
igxShape.Graphic.Replace igxBuilder1.Graphic
Set igxGraphics = igxShape.Graphic.GraphicGroup.Graphics
' Move the Rectangle to the front
MsgBox "Click OK to make the Rectangle draw on top of the ellipse"
igxGraphics.MoveToFront 1
MsgBox "Click OK to continue"
```

The following example creates four shapes, and assigns polygon names to each. Then four polygons are created in a GraphicBuilder, and a Graphics object is set to it. The MoveToFront method is used to move the first polygon to the front, which alters the order of all the polygons in the collection. The polygons on the shapes, and the shapes' labels show the result.

```
THIS MAY CAUSE A FATAL CRASH

Private Sub Main()
    'Dimension subroutine variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxShape3 As Shape
    Dim igxShape4 As Shape
    Dim igxGraphics As Graphics
    Dim igxBuilder1 As New GraphicBuilder
    ' Create 4 new shapes
     Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
    Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 5, 1440)
    Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 7, 1440)
    ' Label the shapes
    igxShape1.Text = "Triangle "
    igxShape2.Text = "Rectangle "
    igxShape3.Text = "Pentagon "
    igxShape4.Text = "Hexagon "
    'Add 4 polygons to the GraphicBuilder
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 3, -90
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 4, 45
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 5, -90
    igxBuilder1.Polygon 0.5, 0.5, 0.5, 6
    'Set the igxGraphics variable to the GraphicBuilder1
    '   Graphics property
    Set igxGraphics = igxBuilder1.Graphic.GraphicGroup.Graphics
    'Move the last item to the back (making it the first item)
    igxGraphics.MoveToFront (1)
    'Replace the graphic in each shape with a polygon
    'from the igxGraphics object
    igxShape1.Graphic.Replace igxGraphics.Item(1)
    igxShape2.Graphic.Replace igxGraphics.Item(2)
    igxShape3.Graphic.Replace igxGraphics.Item(3)
    igxShape4.Graphic.Replace igxGraphics.Item(4)
    'Pause for the user
    MsgBox "Notice the shift in the polygon order."
End Sub
```

{button Graphics object,JI(`igrafxrf.HLP',`Graphics_Object')}

GraphicGroup Object

The GraphicGroup object groups one or more Graphic objects. Its purpose is to allow operations to be performed on several graphic objects at once.

A GraphicGroup can contain many different primitive graphic types. These different graphic types are contained within the group, and as such, this object provides a way to get at the different graphic types in the group.

Properties, Methods, and Events

All of the properties, methods, and events for the GraphicGroup object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Graphics		
Parent		

Graphics Property

Syntax *GraphicGroup*.**Graphics**

Data Type Graphics collection object (read-only, See Object Properties)

Description The Graphics property returns the Graphics collection for the specified GraphicGroup object. The Graphics object is a collection object that contains several single graphic objects.

The Graphics object and GraphicGroup object work together. DiagramObjects do not know how to display a Graphics collection, only a single graphic. The GraphicGroup object works with the Graphics collection to supply a DiagramObject with a single graphic by combining the Graphics collection into a single graphic, before giving it to a diagram object.

Example The following example uses the GraphicGroup.Graphics property to replace the graphic items in the four shapes placed in the diagram with graphic items from the GraphicGroup of the GraphicBuilder.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxShape3 As Shape
   Dim igxShape4 As Shape
   Dim igxGraphicGroup As GraphicGroup
   Dim igxBuilder1 As New GraphicBuilder
   ' Create 4 new shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 3, 1440)
   Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 5, 1440)
   Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 7, 1440)
   ' Label the shapes
   igxShape1.Text = "Triangle "
   igxShape2.Text = "Rectangle "
   igxShape3.Text = "Pentagon "
   igxShape4.Text = "Hexagon "
   MsgBox "View the diagram with the 4 shapes"
   ' Add 4 polygons to the GraphicBuilder
   igxBuilder1.Polygon 0.5, 0.5, 0.5, 3, -90
   igxBuilder1.Polygon 0.5, 0.5, 0.5, 4, 45
   igxBuilder1.Polygon 0.5, 0.5, 0.5, 5, -90
   igxBuilder1.Polygon 0.5, 0.5, 0.5, 6
   ' Get the GraphicGroup object from igxBuilder1
   Set igxGraphicGroup = igxBuilder1.Graphic.GraphicGroup
   ' Replace the graphic of each shape with an item from
   ' the GraphicGroup
   igxShape1.Graphic.Replace igxGraphicGroup.Graphics.Item(1)
   igxShape2.Graphic.Replace igxGraphicGroup.Graphics.Item(2)
   igxShape3.Graphic.Replace igxGraphicGroup.Graphics.Item(3)
   igxShape4.Graphic.Replace igxGraphicGroup.Graphics.Item(4)
   ' Pause for the user
   MsgBox "The graphic of each shape was replaced." _
       & Chr(13) & "Click OK to continue."
```

```
End Sub
```

See Also

Graphic object

Graphics object

iGrafx API Object Hierarchy

{button GraphicGroup object,JI(`igrafxrf.HLP',`GraphicGroup_Object')}

Guideline Object

A guideline is a tool for aligning diagram objects to specific locations on a diagram. Guidelines help in achieving an attractive, organized layout of the objects on a diagram. Any number of guidelines can be placed on a diagram in both the horizontal and vertical directions. The object is derived from and accessed through the Guidelines collection (which is accessed from the Application object).

If the Snap to Guidelines option is turned on, dragging an edge of a diagram object near a guideline causes the side to snap into alignment with the guideline. Dragging the dashed blue cross in the center of the object near a guideline causes its center to snap into alignment. If a user drags more than one selected object, the selected object beneath the pointer snaps to the guideline.

As a developer, you can use the Guideline object to add, remove, and position either horizontal or vertical guidelines on a diagram. These guidelines then can be used for aligning diagram objects, whether they are created programmatically, or by an interactive user.

The following example creates a vertical guideline at one inch on the active diagram, by adding it to the Guidelines collection.

```
' Dimension the variables
Dim igxGLines As Guidelines
Dim igxGLine As Guideline
' Set the Guidelines object from the ActiveDiagram
' to the guidelines variable
Set igxGLines = ActiveDiagram.Guidelines
' Create the guideline object at one inch and vertical
igxGLines.Add 1440, ixGuidelineVertical
```

Properties, Methods, and Events

All of the properties, methods, and events for the Guideline object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
Parent		
Position		
Type		

Related Topics

Guidelines object

iGrafx API Object Hierarchy

Position Property

Syntax *Guideline*.**Position**

Data Type Long (read/write)

Description The Position property specifies the position of a guideline, in either the horizontal or vertical direction. The fixed component of the position, horizontal or vertical, is specified by the Type property. The value of this property is specified in twips (1440 twips = 1 inch).

Example The following example creates a vertical guideline at one inch and then sets the position of the first Guideline object in the collection to two inches.

```
' Dimension the variables
Dim igxGLines As Guidelines
Dim igxGLine As Guideline
' Set the Guidelines object from the ActiveDiagram
' to the guidelines variable
Set igxGLines = ActiveDiagram.Guidelines
' Create the guideline object at one inch and vertical
igxGLines.Add 1440, ixGuidelineVertical
' Get the first guideline from the guidelines collection
Set igxGLine = igxGLines.Item(1)
' Set the position of the Guideline to be two inches
igxGLine.Position = (1440 * 2)
```

See Also Type property

{button Guideline object,JI(`igrafxrf.HLP',`Guideline_Object')}

Type Property

Syntax *Guideline*.**Type**

Data Type IxGuidelineType enumerated constant (read-only)

Description The Type property returns the type, either horizontal or vertical, for a particular guideline.

The IxGuidelineType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	IxGuidelineVertical
1	IxGuidelineHorizontal

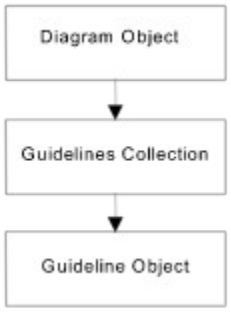
Example The following example creates a vertical guideline at one inch. It then gets the first item from the Guidelines collection and displays a message in the Immediate window indicating whether the guideline is horizontal or vertical.

```
' Dimension the variables
Dim igxGLines As Guidelines
Dim igxGLine As Guideline
' Set the Guidelines object from the ActiveDiagram
' to the guidelines variable
Set igxGLines = ActiveDiagram.Guidelines
' Create the guideline object
igxGLines.Add 1440, ixGuidelineVertical
' Get the first guideline from the guidelines collection
Set igxGLine = igxGLines.Item(1)
' Display the type of the guideline
Select Case igxGLine.Type
   Case ixGuidelineHorizontal:
       MsgBox "Guideline is Horizontal."
   Case ixGuidelineVertical:
       MsgBox "Guideline is Vertical."
End Select
```

{button Guideline object,JI(`igrafxrf.HLP',`Guideline_Object')}

Guidelines Object

Guidelines is a collection of Guideline objects which is derived from and accessed through the Diagram object. Since it is a collection you can Add items to or remove items from the collection. As well as get the count and set the visible state of the entire Guidelines collection.



The following example gets the Guidelines collection from the ActiveDiagram object.

```
' Dimension the variables
Dim igxGLines As Guidelines
' Get the Guidelines collection object
Set igxGLines = ActiveDiagram.Guidelines
```

Properties, Methods, and Events

All of the properties, methods, and events for the Guidelines object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent	RemoveAll	
Visible		

Related Topics

Guideline object

iGrafx API Object Hierarchy

Add Method

Syntax *Guidelines*.**Add**(*Position* As Long, *Type* As IxGuidelineType)

Description The Add method adds a guideline to the Guidelines collection.

The *Position* argument specifies the position at which to place a new guideline on the diagram. It is a single point because the other axis is defined by the Type argument. The argument value is specified in twips.

The *Type* argument specifies the orientation of the guideline. Once the orientation is set it cannot be changed. It can however be verified with the Type property of the Guideline object.

The IxGuidelineType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	IxGuidelineVertical
1	IxGuidelineHorizontal

Example The following example creates a vertical guideline at one inch, by adding it to the Guidelines collection.

```
' Dimension the variables
Dim igxGLines As Guidelines
Dim igxGLine As Guideline
' Set the Guidelines object from the ActiveDiagram
' to the guidelines variable
Set igxGLines = ActiveDiagram.Guidelines
' Create the vertical guideline object
igxGLines.Add 1440, ixGuidelineVertical
```

{button Guidelines object,JI(`igrafxrf.HLP',`Guidelines_Object')}

Item Method

Syntax *Guidelines*.**Item**

Description The Item method returns the Guideline object at the specified *Index* from the Guidelines collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Guideline. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example iterates through the Guidelines collections and displays the postion of each in inches.

```
' Dimension the variables
Dim igxGuideLines As Guidelines
' Get the diagram Guidelines object
Set igxGuideLines = ActiveDiagram.Guidelines
' Add two guidelines
igxGuideLines.Add 2000, ixGuidelineVertical
igxGuideLines.Add 4000, ixGuidelineHorizontal
' Iterate through the Guidelines collection to display the positions
For Index = 1 To igxGuideLines.Count
    If igxGuideLines.Item(Index).Type = ixGuidelineHorizontal Then
        MsgBox "There is a horizontal guideline " & _
            Round(igxGuideLines.Item(Index).Position / 1440, 2) & _
            " inches down."
   End If
   If igxGuideLines.Item(Index).Type = ixGuidelineVertical Then
        MsgBox "There is a vertical guideline " & _
            Round(igxGuideLines.Item(Index).Position / 1440, 2) & _
            " inches over."
   End If
Next Index
```

{button Guidelines object,JI(`igrafxrf.HLP',`Guidelines_Object')}

RemoveAll Method

Syntax *Guidelines*.**RemoveAll**

Description The RemoveAll method removes all of the guidelines from the Guidelines collection.

Example The following example removes all guidelines from the Guidelines collection.

```
' Dimension the variables
Dim igxGLines As Guidelines
' Set the Guidelines object from the ActiveDiagram
' to the guidelines variable
Set igxGLines = ActiveDiagram.Guidelines
' Remove all guidelines from the collection
igxGLines.RemoveAll
```

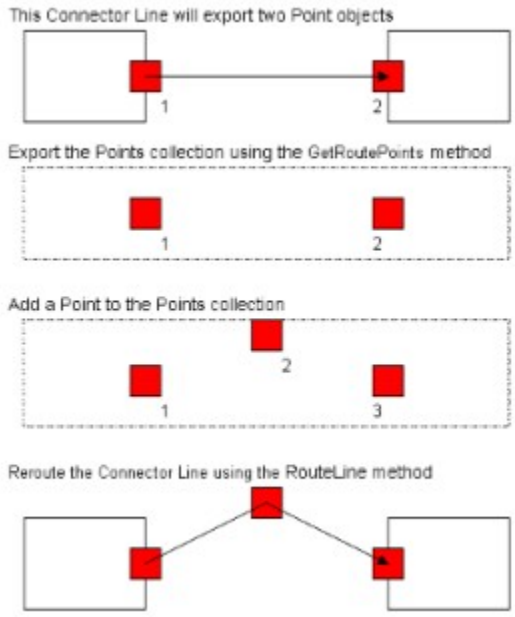
{button Guidelines object,JI(`igrafxrf.HLP',`Guidelines_Object')}

Point Object

The Point object is a routing point for a connector line. A connector line typically has a minimum of two route points (start and end), but may, depending on the routing type (direct, right angle, etc.), have any number of route points.

To access the Points collection for a particular connector line, you use the ConnectorLine.GetRoutePoints method. Then through the connector line's Points collection, you access the indvidual Point objects. A Point object has properties for reading and writing the X and Y positions of the point.

The following illustration shows the effect of adding and moving a new point object, and then importing the Points collection back into the Connector Line.



Note that what you can do with route points may depend on the "routing type" of the connector line.

Properties, Methods, and Events

All of the properties, methods, and events for the Point object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Parent		
X		
Y		

X Property

Syntax *Point*.**X**

Data Type Long (read/write)

Description The X property specifies the position of the specified Point object in the X (horizontal) direction.

Example The following example adds shapes and a connector line to the diagram. It then uses the Point.X and Point.Y properties to number each route point with a text label.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxDiagObj As DiagramObject
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
' Add three shapes, and connect the outer two. The connector
' is routed around the middle shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, _
    ixDirWest)
' Get the connector's Points collection
Set igxPoints = igxConnector.GetRoutePoints
MsgBox "Click OK to number the route points."
' Label all the route points
For Index = 1 To igxPoints.Count
    ActiveDiagram.DiagramObjects.AddTextObject igxPoints _
        .Item(Index).X, igxPoints.Item(Index).Y, , , Str(Index)
Next Index
' Pause for the user
MsgBox "Next, add arrows at each route point. Click OK to continue"
igxConnector.RepeatDestinationArrow = True
MsgBox "There are " & igxPoints.Count & " points in the collection"
MsgBox "Move the third and fourth points closer. Click OK to continue"
MsgBox "Point 3 is at " & igxPoints.Item(3).Y
' Move the Y position of the third and fourth route points
igxPoints.Item(3).Y = 760
igxPoints.Item(4).Y = 760
' Reroute the connector
igxConnector.RouteLine igxPoints
MsgBox "There are " & igxPoints.Count & " points in the collection"
MsgBox "Change the routing type to Direct. Click OK to continue"
igxConnector.Routing = ixRouteDirect
' Get the connector's Points collection
Set igxPoints = igxConnector.GetRoutePoints
' Remove the old route point numbering labels
For Each igxDiagObj In ActiveDiagram.DiagramObjects
    If (igxDiagObj.Type = ixObjectTextGraphic) Then
        igxDiagObj.DeleteDiagramObject
    End If
```

```
Next
MsgBox "There are " & igxPoints.Count & " points in the collection"
MsgBox "Click OK to number the route points."
' Label all the route points
For Index = 1 To igxPoints.Count
    ActiveDiagram.DiagramObjects.AddTextObject igxPoints _
        .Item(Index).X, igxPoints.Item(Index).Y, , , Str(Index)
Next Index
MsgBox "Change the routing type to Lightning bolt. Click OK to continue"
igxConnector.Routing = ixRouteLightningBolt
' Get the connector's Points collection
Set igxPoints = igxConnector.GetRoutePoints
' Remove the old route point numbering labels
For Each igxDiagObj In ActiveDiagram.DiagramObjects
    If (igxDiagObj.Type = ixObjectTextGraphic) Then
        igxDiagObj.DeleteDiagramObject
    End If
Next
MsgBox "There are " & igxPoints.Count & " points in the collection"
MsgBox "Click OK to number the route points."
' Label all the route points
For Index = 1 To igxPoints.Count
    ActiveDiagram.DiagramObjects.AddTextObject igxPoints _
        .Item(Index).X, igxPoints.Item(Index).Y, , , Str(Index)
Next Index
MsgBox "Move the second and third route points. Click OK to continue"
igxPoints.Item(2).X = 1440 * 3
igxPoints.Item(2).Y = 1440 * 1.5
' Reroute the connector
igxConnector.RouteLine igxPoints
MsgBox "View the result"
```

Y Property

Syntax *Point*.**Y**

Data Type Long (read/write)

Description The Y property specifies the position of the specified Point object in the Y (vertical) direction.

Example Refer to the example for the X property .

{button Point object,JI(`igrafxrf.HLP',`Point_Object')}

Points Object

The Points object is a collection of individual Point objects. A Points collection is only associated with the ConnectorLine object. The Points collection can be accessed only by using the ConnectorLine.GetRoutePoints method. The object's purpose is to manipulate a collection of Point objects, which can then be applied to the routing of a Connector Line.

The Points object provides the following functionality:

- The ability to access any Point object in the collection.
- The ability to determine how many Point objects are in the collection.
- The ability to add a new Point object to the collection.

ConnectorLine objects have two important methods for working with a Points collection: *GetRoutePoints*, and *RouteLine*. The GetRoutePoints method exports a Points collection from the Connector Line. The RouteLine method imports a Points collection into a Connector Line.

ConnectorLine objects do not have a Points property. You cannot manipulate the points of a ConnectorLine directly. You must first export the Points collection, and then manipulate it. Changes to the Points collection are totally independent of the source ConnectorLine, and do not affect the ConnectorLine unless you then use the RouteLine method to import the Points back into the ConnectorLine.

The points that determine the routing of a Connector Line are not fixed, they are dynamic. A variety of circumstances, including routing type and the proximity of other objects, can change the routing of a ConnectorLine, and therefore change the points associated with it. For instance, if the ConnectorLine uses the RightAngle routing type, it uses an algorithm for object avoidance which causes the ConnectorLine to detour around other objects, so as not to cross over them. This avoidance changes the points associated with the ConnectorLine, regardless of the Points collection you may supply to it using the RouteLine method. Therefore, the position and number of points on a ConnectorLine may not be entirely predictable.

If the position and number of points on a Connector Line is critical in your diagram, use caution when adding new objects near the ConnectorLine, and when changing objects associated with it. One key aspect to the Points collection is that the order of the Point objects in the collection is important.

Properties, Methods, and Events

All of the properties, methods, and events for the Points object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Clear	
Parent	Item	

Add Method

Syntax *Points*.**Add**(*X* As Long, *Y* As Long) As Point

Description The Add method adds a new Point object to the end of the Points collection. The method returns a Point object. The *X* and *Y* arguments specify the location at which to place the point. The argument values are specified in twips (1440 twips = 1 inch).

A variety of circumstances determine the routing of a ConnectorLine, including routing type and the proximity of other objects. Explicit positioning of a Point in a Points collection does not necessarily force the ConnectorLine to route through that point. For instance, if a ConnectorLine uses *ixRouteFlagFindEdge*, the ConnectorLine ignores the last Point in the Points collection, and connects to the edge of the destination shape, regardless of the position of the last Point in the Points collection (see the example).

Example The following example gets the Points collection from a connector line. One point in the collection is then moved, and another added. Then the connector line is rerouted based on the new Points collection.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
' Add three shapes, and connect the outer two. The connector
' is routed around the middle shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 6, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, _
    ixDirWest)
' Get the connector's Points collection
Set igxPoints = igxConnector.GetRoutePoints
' Add a point to the connector line
MsgBox "Click OK to add one route point to the connector."
' Add a point to the collection
igxPoints.Add 6000, 2000
' Move the last point away from the shape
igxPoints.Item(2).X = 5000
igxPoints.Item(2).Y = 1000
' Import the Points back into the Connector
igxConnector.RouteLine igxPoints
' Reconnect the shapes
igxConnector.ReconnectSource igxShape1, , ixDirEast
igxConnector.ReconnectDestination igxShape2, , ixDirWest
' Label the route points
MsgBox "Click OK to number the route points."
For Index = 1 To igxPoints.Count
    ActiveDiagram.DiagramObjects.AddTextObject igxPoints _
        .Item(Index).X, igxPoints.Item(Index).Y, , , Str(Index)
Next Index
' Pause for the user
MsgBox "Point 3 is ignored in favor of connecting to Shape 2."
```

See Also

ConnectorLine.RouetLine method

ConnectorLine.ReconnectSource method

ConnectorLine.ReconnectDestination method

{button Points object,JI(`igrafxrf.HLP',`Points_Object')}

Clear Method

Syntax *Points*.**Clear**

Description The Clear method removes all of the Point objects from the collection. The Clear method provides a way to empty a Points collection so that a custom Points collection can be designed and applied to a connector line.

A variety of circumstances determine the routing of a ConnectorLine, including routing type and the proximity of other objects. Explicit positioning of a Point in a Points collection does not necessarily force the ConnectorLine to route through that point. For instance, if a ConnectorLine uses *ixRouteFlagFindEdge*, the ConnectorLine ignores the last Point in the Points collection, and connects to the edge of the destination shape, regardless of the position of the last Point in the Points collection (see the example).

Example The following example retrieves the Points collection from a connector line, clears it, constructs a new Points collection, and reapplies the new collection to the connector line.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
' Add two shapes, and connect them
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, _
    ixDirWest)
' Get the connector's Points collection
Set igxPoints = igxConnector.GetRoutePoints
' Clear all points from the collection
MsgBox "Click OK to clear the route points."
igxPoints.Clear
' Add three new points to igxPoints
MsgBox "Click OK to add two new route points."
igxPoints.Add 1440 * 2, 1440
igxPoints.Add 1440 * 4, 1440
' Reroute the connector with the points
igxConnector.RouteLine igxPoints, ixRouteFlagDontFindEdge
' Pause for the user
MsgBox "Click OK to continue."
```

{button Points object,JI(`igrafxrf.HLP',`Points_Object')}

Item Method

Syntax *Points*.**Item**(*Index* As Integer) As Point

Description The Item method returns the Point object at the specified *Index* from the Points collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Point. An error is produced if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example adds shapes and a connector line to the diagram. It then uses the Point.X and Point.Y properties of the Point objects' to number each route point with a text label.

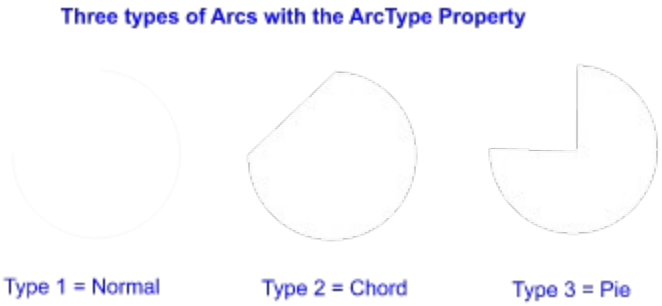
```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
' Add three shapes, and connect the outer two. The connector
' is routed around the middle shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, _
    ixDirWest)
' Get the connector's Points collection
Set igxPoints = igxConnector.GetRoutePoints
MsgBox "Click OK to number the route points."
' Label all the route points
For Index = 1 To igxPoints.Count
   ActiveDiagram.DiagramObjects.AddTextObject igxPoints _
       .Item(Index).X, igxPoints.Item(Index).Y, , , Str(Index)
Next Index
' Pause for the user
MsgBox "Click OK to continue"
```

{button Points object,JI(`igrafxrf.HLP',`Points_Object')}

ArcGraphic Object

The ArcGraphic object is one of several graphic primitives available to iGrafx Professional users. As the name implies, this object controls arc-shaped graphics. The ArcGraphic object is subordinate to the Graphic object, and can be accessed only through the Graphic object when the Graphic.Type property is equal to ixGraphicArc.

There are three types of arc that the ArcGraphic object can contain, as shown in the following illustration. The type is designated by the ArcType property.



The remaining properties of this object allow you to size and position the arc. All of theses arguments have their counterpart as arguments to the GraphicBuilder.Arc method.

Properties, Methods, and Events

All of the properties, methods, and events for the ArcGraphic object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
ArcType		
Bottom		
EndX		
EndY		
Left		
Parent		
Right		
StartX		
StartY		
Top		

Related Topics

GraphicBuilder object

ArcType Property

Syntax *ArcGraphic*.**ArcType**

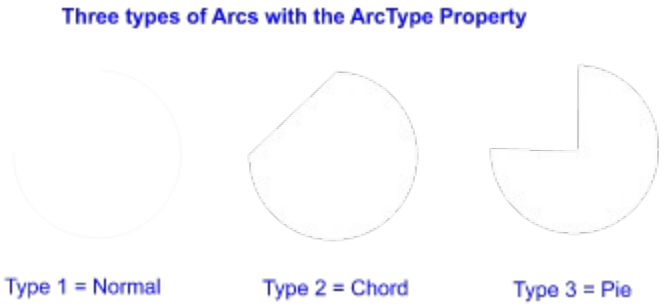
Data Type IxArcType enumerated constant (read/write)

Description The ArcType property specifies the type of arc that is defined by the ArcGraphic object. If you initially draw an arc using one type, you also can change the type of arc later with this property.

The IxArcType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixArcNormal
1	ixArcChord
2	ixArcPie

The following illustration shows the types of arc that are defined by each of the IxArcType constants. You can create any of these arc types with the various methods of the GraphicBuilder object. Similarly, you could use the GraphicBuilder.Arc method to draw a “normal” arc, and use this property to change it to either a chord- or pie-type arc.



Example The following example uses a GraphicBuilder to draw an arc. A new shape is placed on the diagram, and it's graphic is replaced with the arc. The ArcType is then changed to a Pie.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBuilder As New GraphicBuilder
Dim igxGraphic As Graphic
Dim igxArc As ArcGraphic
' Use the GraphicBuilder to draw an arc
igxBuilder.Arc 0, 0, 1, 1, 0.2, 0.8, 0.8, 0.8, True
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 + 360, 1440 + 360)
' Make the shape larger
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
' Replace the shape's graphic with the arc
igxShape.Graphic.Replace igxBuilder.Graphic
' Change the ArcType to a Pie
MsgBox "Click OK to change the arc to a Pie."
igxShape.Graphic.ArcGraphic.ArcType = ixArcPie
' Pause for the user
MsgBox "Click OK to continue."
```

See Also GraphicBuilder.Arc method

{button ArcGraphic object,JI(`igrafxrf.HLP',`ArcGraphic_Object')}

Bottom Property

Syntax *ArcGraphic*.**Bottom**

Data Type Double (read/write)

Description The Bottom property specifies the lowest point within the arc's bounding rectangle to which the arc is drawn. No point on the arc curve can drop below the Bottom property's value. The value is expressed in terms of the relative coordinate space (that is, the bounding box) of the arc, and is usually a value between 0.0 and 1.0.

Example The following example draws a large arc, and then points to each of the Bottom, Top, Left, Right, Start, and End properties on screen, and shows the value of each.

```
Private Sub Main()
    ' Dimension subroutine variables
   Dim igxShape As Shape
   Dim igxBuilder As New GraphicBuilder
   Dim igxGraphic As Graphic
   Dim igxArc As ArcGraphic
   ' Use the GraphicBuilder to draw an arc
   igxBuilder.Arc 0, 0, 1, 1, 0.2, 0.8, 0.8, 0.8, True
   ' Add a shape to the diagram
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
   ' Make the shape larger
   igxShape.DiagramObject.Width = 1440 * 2
   igxShape.DiagramObject.Height = 1440 * 2
   ' Replace the shape's graphic with the arc
   igxShape.Graphic.Replace igxBuilder.Graphic
   ' Get the shape's new ArcGraphic
   Set igxArc = igxShape.Graphic.ArcGraphic
   ' Point to the ArcGraphic properties
   ' Select the shape
   igxShape.DiagramObject.Selected = True
   PointTo igxShape, igxArc.StartX, igxArc.StartY, "Start"
   ' Select the shape
   igxShape.DiagramObject.Selected = True
   PointTo igxShape, igxArc.EndX, igxArc.EndY, "End"
   ' Select the shape
   igxShape.DiagramObject.Selected = True
   PointTo igxShape, igxArc.Left, igxArc.Top, "Left, Top"
   ' Select the shape
   igxShape.DiagramObject.Selected = True
   PointTo igxShape, igxArc.Right, igxArc.Bottom, "Right, Bottom"
End Sub

' This subroutine does the pointing
Private Sub PointTo(Shape As Shape, X As Double, Y As Double, Label As String)
   ' Dimension the variables
   Dim igxConnector As ConnectorLine
   Dim igxText As TextGraphicObject
   Dim ArcBottom, ArcTop, ArcLeft, ArcRight As Long
   Dim Height, Width As Long
   ' Get the shape's coordinates
   ArcBottom = Shape.DiagramObject.Bottom
```

```
    ArcTop = Shape.DiagramObject.Top
    ArcLeft = Shape.DiagramObject.Left
    ArcRight = Shape.DiagramObject.Right
    Height = Shape.DiagramObject.Height
    Width = Shape.DiagramObject.Width
    ' Draw a connector line that points to the property
    Select Case Label
        Case "Start":
            Set igxConnector = ActiveDiagram.DiagramObjects. _
                AddConnectorLine(ixRouteDirect, _
                ixRouteFlagDontFindEdge, , , , 5500, 5500, , , , _
                ArcRight - (Width * 0.15), ArcBottom - (Height * 0.15))
        Case "End":
            Set igxConnector = ActiveDiagram.DiagramObjects. _
                AddConnectorLine(ixRouteDirect, _
                ixRouteFlagDontFindEdge, , , , 5500, 5500, , , , _
                ArcLeft + (Width * 0.15), ArcBottom - (Height * 0.15))
        Case "Left, Top":
            Set igxConnector = ActiveDiagram.DiagramObjects. _
                AddConnectorLine(ixRouteDirect, _
                ixRouteFlagDontFindEdge, , , , 5500, 5500, , , , _
                ArcLeft + 25, ArcTop - 25)
        Case "Right, Bottom":
            Set igxConnector = ActiveDiagram.DiagramObjects. _
                AddConnectorLine(ixRouteDirect, _
                ixRouteFlagDontFindEdge, , , , 5500, 5500, , , , _
                ArcRight + 25, ArcBottom - 25)
    End Select
    ' Display the label text at the foot of the arrow
    Set igxText = ActiveDiagram.DiagramObjects.AddTextObject _
        (5500, 5500, , , Label & " " & X & ", " & Y)
    MsgBox "Click OK to continue"
    ' Remove the pointing graphics
    igxConnector.Delete
    igxText.DiagramObject.DeleteDiagramObject
End Sub
```

See Also

Left property

Right property

Top property

{button ArcGraphic object,JI(`igrafxrf.HLP',`ArcGraphic_Object')}

EndX Property

Syntax *ArcGraphic*.**EndX**

Data Type Double (read/write)

Description The EndX property, along with the EndY property, specifies the position inside the bounding box of the arc where the arc curve ends. The value is expressed in terms of the relative coordinate space, and is usually a value between 0.0 and 1.0. Because the arc is a curve, the end of the line curve may not fall exactly on the EndX and EndY position. The actual curve is drawn as close as possible.

Example Refer to the example for the Bottom property.

See Also EndY property

StartX property

StartY property

{button ArcGraphic object,JI(`igrafxrf.HLP',`ArcGraphic_Object')}

EndY Property

Syntax *ArcGraphic*.**EndY**

Data Type Double (read/write)

Description The EndY property, along with the EndX property, specifies the position inside the bounding box of the arc where the arc curve ends. The value is expressed in terms of the relative coordinate space, and is usually a value between 0.0 and 1.0. Because the arc is a curve, the end of the line curve may not fall exactly on the EndX and EndY position. The actual curve is drawn as close as possible.

Example Refer to the example for the Bottom property.

See Also EndX property

StartX property

StartY property

{button ArcGraphic object,JI(`igrafxrf.HLP',`ArcGraphic_Object')}

Left Property

Syntax *ArcGraphic*.**Left**

Data Type Double (read/write)

Description The Left property specifies the left-most point within the arc's bounding box to which the arc curve is drawn. No point on the arc curve can be drawn further left than this value. The value is expressed in terms of the relative coordinate space, which is usually a value between 0.0 and 1.0.

Example Refer to the example for the Bottom property.

See Also Bottom property

Right property

Top property

{button ArcGraphic object,JI(`igrafxrf.HLP',`ArcGraphic_Object')}

Right Property

Syntax *ArcGraphic*.**Right**

Data Type Double (read/write)

Description The Right property specifies the right-most point within the arc's bounding box to which the arc curve is drawn. No point on the arc curve can be drawn further right than this value. The value is expressed in terms of the relative coordinate space, which is usually a value between 0.0 and 1.0.

Example Refer to the example for the Bottom property.

See Also Bottom property

Left property

Top property

{button ArcGraphic object,JI(`igrafxrf.HLP',`ArcGraphic_Object')}

StartX Property

Syntax *ArcGraphic*.**StartX**

Data Type Double (read/write)

Description The StartX property, along with the StartY property, specifies the position inside the bounding box of the arc where the arc curve begins. The value is expressed in terms of the relative coordinate space, and is usually a value between 0.0 and 1.0. Because the arc is a curve, the end of the line curve may not fall exactly on the EndX and EndY position. The actual curve is drawn as close as possible.

Example Refer to the example for the Bottom property.

See Also EndX property

EndY property

StartY property

{button ArcGraphic object,JI(`igrafxrf.HLP',`ArcGraphic_Object')}

StartY Property

Syntax *ArcGraphic*.**StartY**

Data Type Double (read/write)

Description The StartY property, along with the StartX property, specifies the position inside the bounding box of the arc where the arc curve begins. The value is expressed in terms of the relative coordinate space, and is usually a value between 0.0 and 1.0. Because the arc is a curve, the end of the line curve may not fall exactly on the EndX and EndY position. The actual curve is drawn as close as possible.

Example Refer to the example for the Bottom property.

See Also EndX property

EndY property

StartX property

{button ArcGraphic object,JI(`igrafxrf.HLP',`ArcGraphic_Object')}

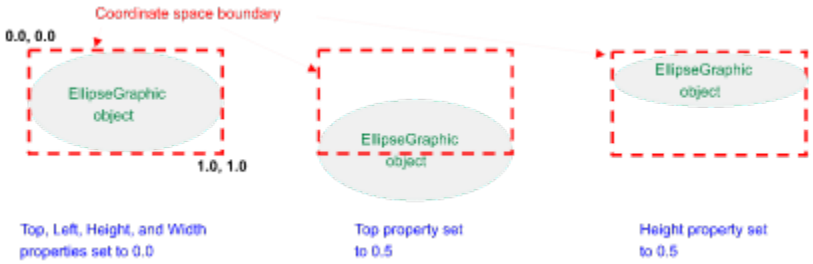
EllipseGraphic Object

The EllipseGraphic object is one of several graphic primitives available to iGrafx Professional users. As the name implies, this object controls elliptically-shaped graphics. The EllipseGraphic object is subordinate to the Graphic object, and can be accessed only through the Graphic object.

The EllipseGraphic property allows the programmer to access the EllipseGraphic object level in order to read or write any of its properties. To create an ellipse, use the GraphicBuilder object.

The properties of the ellipse, such as Left and Top, are relative to the coordinate space of the shape.

The following diagrams illustrate how the properties work for the EllipseGraphic object. Assume that the ellipse is a Shape object.



In the diagram, the dashed red line shows the coordinate space boundary of the Shape object, which ranges from 0.0 to 1.0 in both X and Y. The Width and Left properties are not shown in the diagram because their effect is the same as the Height and Top properties (except they work in the X direction). The size of the ellipse is controlled by the Height and Width properties. The size is always relative to the coordinate space. Note that in the case of the Top property being set at 0.5, if a user clicks on the area of the ellipse that is outside the coordinate space, the mouse click has no effect (the coordinate space, at the Shape level, is what triggers events such as mouse clicks).

Properties, Methods, and Events

All of the properties, methods, and events for the EllipseGraphic object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Height		
Left		
Parent		
Top		
Width		

Related Topics

GraphicBuilder object

Left Property

Syntax *EllipseGraphic*.**Left**

Data Type Double (read/write)

Description The Left property controls the position of the left side of an ellipse relative to the coordinate space of the parent object (in the case of all graphic primitives, this ends up being either a Shape, ShapeClass, ShapeLibrary, or TextGraphicObject object). The units for this property are typically between 0.0 and 1.0; however, coordinate spaces are not required to be between zero and one—they can be re-defined. The range of values may be different if the coordinate space of the shape was intentionally modified.

Example The following example creates an ellipse with the GraphicBuilder object and places a shape in the active diagram. It then replaces the graphic of the shape with the new ellipse from the GraphicBuilder. Then the example shows the effect of adjusting the Left, Top, Height, and Width properties.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBuilder As New GraphicBuilder
Dim igxGraphic As Graphic
Dim igxEllipse As EllipseGraphic
' Use the GraphicBuilder to draw an ellipse
igxBuilder.Ellipse 0, 0, 1, 1
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
MsgBox "View the diagram"
' Replace the shape's graphic with the ellipse
igxShape.Graphic.Replace igxBuilder.Graphic
MsgBox "Replaced the shape's graphic with an ellipse"
' Make the shape larger
igxShape.DiagramObject.Width = 1440 * 2.5
igxShape.DiagramObject.Height = 1440 * 1.5
MsgBox "Made the shape larger"
' Select the shape
igxShape.DiagramObject.Selected = True
' Get the shape's new EllipseGraphic
Set igxEllipse = igxShape.Graphic.EllipseGraphic
' Adjust the Left property of the ellipse
MsgBox "Set ellipse Left property to 0.3; Click OK"
igxEllipse.Left = 0.3
MsgBox "Set ellipse Left property to -0.3; Click OK"
igxEllipse.Left = -0.3
' Adjust the Top property of the ellipse
MsgBox "Set Left back to 0, and set Top to 0.5; Click OK"
igxEllipse.Left = 0
igxEllipse.Top = 0.5
MsgBox "Set ellipse Top property to -0.5; Click OK"
igxEllipse.Top = -0.5
' Adjust the Height property of the ellipse
MsgBox "Set Top back to 0, and set Height to 0.7; Click OK"
igxEllipse.Top = 0
igxEllipse.Height = 0.7
MsgBox "Set ellipse Height property to -0.7; Click OK"
```

```
igxEllipse.Height = -0.7
' Adjust the Width property of the ellipse
MsgBox "Set Height back to 1, and set Width to 0.6; Click OK"
igxEllipse.Height = 1
igxEllipse.Width = 0.6
MsgBox "Set ellipse Width property to -0.6; Click OK"
igxEllipse.Width = -0.6
MsgBox "Done with example"
```

See Also

Top property

Height property

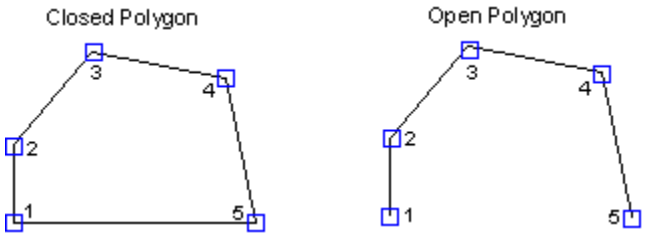
Width property

{button EllipseGraphic object,JI(`igrafxrf.HLP',`EllipseGraphic_Object')}

PolygonGraphic Object

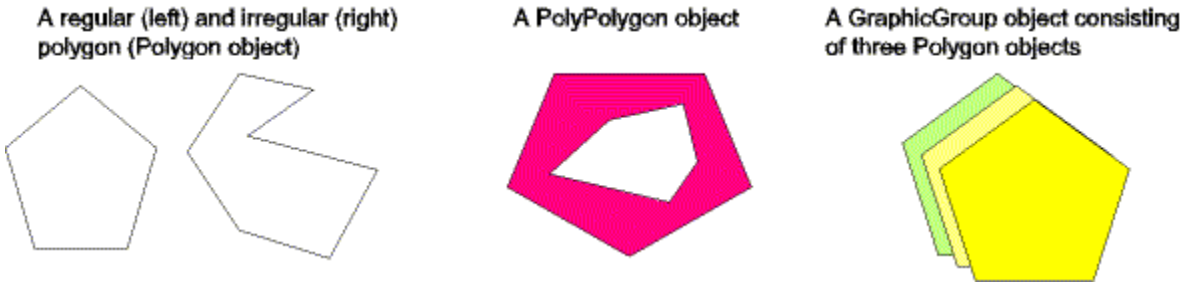
The PolygonGraphic object is one of several graphic primitives available to iGrafx Professional users. As the name implies, this object controls polygonally-shaped graphics. The PolygonGraphic object is subordinate to the Graphic object, and can be accessed only through the Graphic object.

The following diagram describes a polygon from the developer's perspective. The blue rectangles indicate the polygon points. The numbers indicate the order of creation of the polygon points, and are how you refer to a specific polygon point.



A polygon can be created using the GraphicBuilder object. A polygon is described by the locations of its points, and whether it is open or closed.

Polygons are the most flexible of all the graphic primitives in iGrafx Professional. In fact, iGrafx Professional provides several ways of creating polygonally-shaped objects (see also, PolyPolygon and GraphicGroup objects). The following diagram illustrates the differences in these three objects.



For more information about the graphical shapes you can draw with iGrafx Professional, refer to the iGrafx Professional User's Guide.

Properties, Methods, and Events

All of the properties, methods, and events for the PolygonGraphic object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Closed		
Parent		
PolygonPoints		

Related Topics

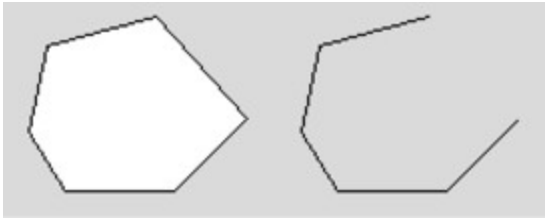
GraphicBuilder object

Closed Property

Syntax *PolygonGraphic*.**Closed**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Closed property specifies whether a polygon is closed or open. If set to True, the final segment of a Polygon graphic is drawn automatically, connecting the end point to the start point, and the Polygon appears closed. If set to False, the final segment of the Polygon is not drawn, leaving a gap between the start and end points, and the Polygon appears open. A closed Polygon can be filled. An open Polygon cannot be filled with a fill color or pattern.



Closed = True Closed = False

Example The following example first creates a polygon using the GraphicBuilder.Polygon method. It then adds a shape in the active diagram, and replaces the shape's graphic with the polygon. After resizing the shape, the PolygonGraphic.Closed method is used to open and close the polygon. Then, to show that PolygonGraphic objects can be created by drawing line segments, the GraphicBuilder object's LineTo method is used to draw an open polygon. Another shape is created and its graphic replaced with the new open polygon. Again the Closed method is used to close and then open the polygon.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBuilder As New GraphicBuilder
Dim igxGraphic As Graphic
Dim igxPolygon As PolygonGraphic
' Use the GraphicBuilder to draw a polygon
igxBuilder.Polygon 0.5, 0.5, 1, 7, 0
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
MsgBox "View the diagram"
' Replace the shape's graphic with the polygon
igxShape.Graphic.Replace igxBuilder.Graphic
MsgBox "Replaced the shape's graphic with a polygon"
' Make the shape larger
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
MsgBox "Made the shape larger"
' Get the shape's new PolygonGraphic
Set igxPolygon = igxShape.Graphic.PolygonGraphic
MsgBox "The polygon is closed. Click OK to make it an open polygon"
igxPolygon.Closed = False
MsgBox "The polygon is now open. Click OK to make it closed again"
igxPolygon.Closed = True
MsgBox "View the diagram"
' Reduce the shape's size and reposition it
igxShape.DiagramObject.Width = 1440
igxShape.DiagramObject.Height = 1440
```

```
igxShape.DiagramObject.CenterX = 1440
igxShape.DiagramObject.CenterY = 1440
MsgBox "Resized and moved the shape. Click OK to create a " _
    & "new polygon by drawing line segments."
' Create a new polygon by drawing line segments
Set igxBuilder = New GraphicBuilder
igxBuilder.BeginPath
igxBuilder.MoveTo 0.3, 0
igxBuilder.LineTo 0.7, 0
igxBuilder.LineTo 1, 0.3
igxBuilder.LineTo 1, 0.7
igxBuilder.LineTo 0.7, 1
igxBuilder.LineTo 0.3, 1
igxBuilder.LineTo 0, 0.7
igxBuilder.EndPath
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 4)
MsgBox "View the diagram"
' Replace the shape's graphic with the polygon
igxShape.Graphic.Replace igxBuilder.Graphic
MsgBox "Replaced the shape's graphic with a polygon"
' Make the shape larger
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
MsgBox "Made the shape larger"
' Get the shape's new PolygonGraphic
Set igxPolygon = igxShape.Graphic.PolygonGraphic
MsgBox "The polygon is open. Click OK to make it a closed polygon"
igxPolygon.Closed = True
MsgBox "The polygon is now closed. Click OK to make it open again"
igxPolygon.Closed = False
MsgBox "Done with example"
```

PolygonPoints Property

Syntax *PolygonGraphic*.**PolygonPoints**

Data Type PolygonPoints object (read-only, See Object Properties)

Description The PolygonPoints property returns the PolygonPoints collection object for the specified polygon. The PolygonPoints collection allows the programmer to access the individual points of a polygon.

Example The following example creates a polygon with the GraphicBuilder, and then replaces the graphic of a shape with the polygon. Then the polygon's PolygonPoints collection is accessed, and the locations of the polygon points are displayed in a message box. Next, the shape is made larger, and the polygon point locations are again displayed. Note that making the shape larger did not affect the polygon point locations because they use the shape's relative coordinate space.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBuilder As New GraphicBuilder
Dim igxGraphic As Graphic
Dim igxPolygonPoints As PolygonPoints
' Use the GraphicBuilder to draw a polygon
igxBuilder.Polygon 0.5, 0.5, 1, 7, 0
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
MsgBox "View the diagram"
' Replace the shape's graphic with the polygon
igxShape.Graphic.Replace igxBuilder.Graphic
MsgBox "Replaced the shape's graphic with a polygon"
' Get the PolygonPoints collection
Set igxPolygonPoints = igxShape.Graphic.PolygonGraphic.PolygonPoints
' Display number of points and their locations
MsgBox "The polygon consists of " & igxPolygonPoints.Count _
    & " points."
' Select the shape
igxShape.DiagramObject.Selected = True
For iCount = 1 To igxPolygonPoints.Count
    sPointLocations = sPointLocations & "Point " & iCount _
        & " is at X = " & igxPolygonPoints.Item(iCount).X _
        & " and Y = " & igxPolygonPoints.Item(iCount).Y & Chr(13)
Next iCount
MsgBox "The locations are: " & Chr(13) & sPointLocations
' Make the shape larger
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
MsgBox "Made the shape larger. Get the point locations again."
sPointLocations = ""
' Display location of points again
For iCount = 1 To igxPolygonPoints.Count
    sPointLocations = sPointLocations & "Point " & iCount _
        & " is at X = " & igxPolygonPoints.Item(iCount).X _
        & " and Y = " & igxPolygonPoints.Item(iCount).Y & Chr(13)
Next iCount
MsgBox "The locations are: " & Chr(13) & sPointLocations
MsgBox "Done with example"
```

See Also

PolygonPoints object

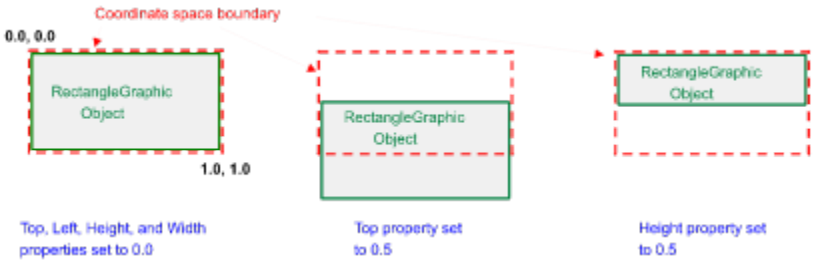
iGrafx API Object Hierarchy

{button PolygonGraphic object,JI(`igrafxrf.HLP',`PolygonGraphic_Object')}

RectangleGraphic Object

The RectangleGraphic object is one of several graphic primitives available to iGrafx Professional users. As the name implies, this object controls rectangularly-shaped graphics. The RectangleGraphic object is subordinate to the Graphic object, and can be accessed only through the Graphic object.

The following diagrams illustrate how the properties work for the RectangleGraphic object. Assume that the rectangle is the graphic being used for a Shape object.



In the diagram, the dashed red line shows the coordinate space boundary of the Shape object, which ranges from 0.0 to 1.0 in both X and Y. The Width and Left properties are not shown in the diagram because their effect is the same as the Height and Top properties (except they work in the X direction). The size of the rectangle is controlled by the Height and Width properties. The size is always relative to the coordinate space. Note that in the case of the Top property being set at 0.5, if a user clicks on the area of the rectangle that is outside the coordinate space, the mouse click has no effect (the coordinate space, at the Shape level, is what triggers events such as mouse clicks).

To create a RectangleGraphic primitive, use the GraphicBuilder object.

Properties, Methods, and Events

All of the properties, methods, and events for the RectangleGraphic object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Height		
Left		
Parent		
Rounding		
Top		
Width		

Left Property

Syntax *RectangleGraphic*.**Left**

Data Type Double (read/write)

Description The Left property controls the position of the left side of a rectangle relative to the coordinate space of the parent object (in the case of all graphic primitives, this ends up being either a Shape, ShapeClass, ShapeLibrary, or TextGraphicObject object). The units for this property are typically between 0.0 and 1.0; however, coordinate spaces are not required to be between zero and one—they can be re-defined. The range of values may be different if the coordinate space of the shape was intentionally modified.

Example The following example creates a rectangle with the GraphicBuilder object and places a shape in the active diagram. It then replaces the graphic of the shape with the new rectangle from the GraphicBuilder. Then the example shows the effect of adjusting the Left, Top, Height, and Width properties.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBuilder As New GraphicBuilder
Dim igxGraphic As Graphic
Dim igxRectangle As RectangleGraphic
' Use the GraphicBuilder to draw an rectangle
igxBuilder.Rectangle 0, 0, 1, 1
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
MsgBox "View the diagram"
' Replace the shape's graphic with the rectangle
igxShape.Graphic.Replace igxBuilder.Graphic
MsgBox "Replaced the shape's graphic with an rectangle"
' Make the shape larger
igxShape.DiagramObject.Width = 1440 * 2.5
igxShape.DiagramObject.Height = 1440 * 1.5
MsgBox "Made the shape larger"
' Select the shape
igxShape.DiagramObject.Selected = True
' Get the shape's new RectangleGraphic
Set igxRectangle = igxShape.Graphic.RectangleGraphic
' Adjust the Left property of the rectangle
MsgBox "Set rectangle Left property to 0.3; Click OK"
igxRectangle.Left = 0.3
MsgBox "Set rectangle Left property to -0.3; Click OK"
igxRectangle.Left = -0.3
' Adjust the Top property of the rectangle
MsgBox "Set Left back to 0, and set Top to 0.5; Click OK"
igxRectangle.Left = 0
igxRectangle.Top = 0.5
MsgBox "Set rectangle Top property to -0.5; Click OK"
igxRectangle.Top = -0.5
' Adjust the Height property of the rectangle
MsgBox "Set Top back to 0, and set Height to 0.7; Click OK"
igxRectangle.Top = 0
igxRectangle.Height = 0.7
MsgBox "Set rectangle Height property to -0.7; Click OK"
```

```
igxRectangle.Height = -0.7
' Adjust the Width property of the rectangle
MsgBox "Set Height back to 1, and set Width to 0.6; Click OK"
igxRectangle.Height = 1
igxRectangle.Width = 0.6
MsgBox "Set rectangle Width property to -0.6; Click OK"
igxRectangle.Width = -0.6
MsgBox "Done with example"
```

{button RectangleGraphic object,JI(`igrafxrf.HLP',`RectangleGraphic_Object')}

Rounding Property

Syntax *RectangleGraphic*.**Rounding**

Data Type Double (read/write)

Description The Rounding property rounds the corners of the specified RectangleGraphic object. The units of the property are specified in twips (1440 twips = 1 inch). A rounded corner is essentially a quarter circle. The Rounding property determines the radius of the quarter circle used to draw the corner.

Overlapping rounded corners are not allowed. Rounded corners are drawn only as large as the rectangle can accommodate; that is, the rectangle corners are rounded at a maximum to 50% of the size of the shortest size if theis property specifies a greater rounding value.

If the rectangle is forced to accommodate rounded corners smaller than the Rounding property specifies, the Rounding property is not changed. It retains the specified value, and if the rectangle is later changed to a larger size, the rounded corners grow to accommodate the value of the property.

Example The following example creates a RectangleGraphic, and then steps through various values for the Rounding property.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBuilder As New GraphicBuilder
Dim igxGraphic As Graphic
Dim igxRectangle As RectangleGraphic
' Use the GraphicBuilder to draw an rectangle
igxBuilder.Rectangle 0, 0, 1, 1
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
MsgBox "View the diagram"
' Replace the shape's graphic with the rectangle
igxShape.Graphic.Replace igxBuilder.Graphic
MsgBox "Replaced the shape's graphic with an rectangle"
' Make the shape larger
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Height = 1440 * 2
MsgBox "Made the shape larger"
' Select the shape
igxShape.DiagramObject.Selected = True
' Get the shape's new RectangleGraphic
Set igxRectangle = igxShape.Graphic.RectangleGraphic
MsgBox "Rounding is at " & igxRectangle.Rounding
' Adjust the Rounding property of the rectangle
For RadiusInTwips = 360 To 4320 Step 360
    igxRectangle.Rounding = RadiusInTwips
    MsgBox "Rounding set to " & igxRectangle.Rounding
Next RaduisInTwips
```

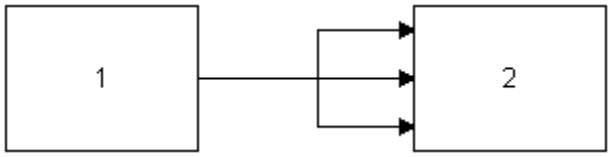
{button RectangleGraphic object,JI(`igrafxrf.HLP',`RectangleGraphic_Object')}

PolyPolygonGraphic Object

The PolyPolygonGraphic object provides a method of creating a special type of polygon graphic: those that can have cut-outs. Although this type of graphic could be created with the GraphicGroup object, the PolyPolygon object performs certain actions automatically, and alternates the effect as additional polygons are added.

A PolyPolygonGraphic object is a collection of Polygon objects—it cannot contain other graphic primitive types. The order of the Polygon objects in the collection is important because of the layering (or intersection) effect.

Polygons are the most flexible of all the graphic primitives in iGrafx Professional. In fact, iGrafx Professional provides several ways of creating polygonally-shaped objects (see also, Polygon and GraphicGroup objects). The following diagram illustrates the differences in these three objects.



Consider the PolyPolygon and GraphicGroup objects in the previous diagram. It would seem that you could make the same shape as the PolyPolygon with a GraphicGroup, but that isn't quite true—there is a difference. The actual shape for the PolyPolygon is only the red region. The white region is truly a 'hole' in the graphic. If the same shape is made using a GraphicGroup object, then the white region would still be part of the shape.

The graphical shapes themselves are drawn as polygons, and accessed as Polygon objects. Except for the end result of drawing the member polygons, the PolyPolygonGraphic object acts just like any other collection object within iGrafx Professional.

Properties, Methods, and Events

All of the properties, methods, and events for the PolyPolygonGraphic object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	AddPolygon	
Count	Item	
Parent		

Related Topics

PolygonGraphic object
GraphicGroup object
GraphicBuilder object

AddPolygon Method

Topic Under Construction!!!

Syntax *PolyPolygonGraphic*.**AddPolygon**

Description The AddPolygon method

Example The following code illustrates how to use the AddPolygon method

THIS DOES NOT WORK WORTH A DARN. CONFUSING!!! WHY IS THERE NO ARGUMENT TO THE ADDPOLYGON METHOD? WHAT IS THIS FOR?

```
' Dimension the variables
Dim igxShape As Shape
Dim igxBuilder As New GraphicBuilder
Dim igxGraphic As Graphic
Dim igxGraphicGrp As GraphicGroup
Dim igxPolygon As PolygonGraphic
Dim igxPolyPolygon As PolyPolygonGraphic
' Use the GraphicBuilder to draw two polygons with lines to make
' a PolyPolygonGraphic
igxBuilder.BeginPath
igxBuilder.MoveTo 0, 0
igxBuilder.LineTo 0.5, 0
igxBuilder.LineTo 1, 0.25
igxBuilder.LineTo 0.75, 1
igxBuilder.LineTo 0.5, 1
igxBuilder.Close
igxBuilder.MoveTo 0.5, 0.25
igxBuilder.LineTo 0.75, 0.3
igxBuilder.LineTo 0.75, 0.7
igxBuilder.LineTo 0.5, 0.8
igxBuilder.Close
igxBuilder.EndPath
Set igxGraphic = igxBuilder.Graphic
MsgBox igxGraphic.Type
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Replace the shape's graphic with the polygon
igxShape.Graphic.Replace igxGraphic
' Check the PolyPolygonGraphic collection to see what is contains
MsgBox "PolyPolygon collection has " & igxGraphic.PolyPolygonGraphic.Count & "
item(s)."
igxShape.FillColor = vbRed
MsgBox "View the diagram"
' Resize the shape
igxShape.DiagramObject.Height = 1440 * 2
igxShape.DiagramObject.Width = 1440 * 2
igxShape.DiagramObject.Selected = True
MsgBox "View the diagram"
'Set igxBuilder = New GraphicBuilder
igxBuilder.BeginPath
```

```
igxBuilder.MoveTo 0.7, 0.3
igxBuilder.LineTo 0.7, 0.9
igxBuilder.LineTo 0.4, 0.7
igxBuilder.LineTo 0.2, 0.4
igxBuilder.Close
igxBuilder.EndPath
igxShape.Graphic.PolyPolygonGraphic.AddPolygon
MsgBox "PolyPolygon collection has " & igxShape.Graphic. _
    PolyPolygonGraphic.Count & " item(s)."
```

{button PolyPolygonGraphic object,JI(`igrafxrf.HLP',`PolyPolygonGraphic_Object')}

Item Method

Topic Under Construction!!!

Syntax *PolyPolygonGraphic*.**Item**(*Index* As Integer) As PolygonGraphic

Description The Item method returns the PolygonGraphic object at the specified *Index* from the PolyPolygonGraphic collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type PolygonGraphic. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example

{button PolyPolygonGraphic object,JI(`igrafxrf.HLP',`PolyPolygonGraphic_Object')}

PolygonPoint Object

The PolygonPoint object represents the points that make up a polygon graphic. In the following illustration, the polygon points are indicated by the larger squares at various positions along the line that describes the shape of the graphic.

The PolygonPoint object is subordinate to the Polygon and PolyPolygon objects, by means of the PolygonPoints collection object. Each individual point is kept in the PolygonPoints collection object. Access to the individual polygon points must be done through the PolygonPoints collection.

Each polygon point can be positioned by specifying the X and Y locations for the point. In addition, each PolygonPoint object has a BezierControl object associated with it. The BezierControl object allows you to manipulate the bends of the line segments that enter and exit the polygon point.

Properties, Methods, and Events

All of the properties, methods, and events for the PolygonPoint object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Set	
BezierControl	Smooth	
Parent	Unsmooth	
X		
Y		

BezierControl Property

Syntax *PolygonPoint*.**BezierControl**

Data Type BezierControl object (Read-Only, See Object Properties)

Description The BezierControl property returns the BezierControl object for the specified PolygonPoint. The BezierControl object is used to adjust the curvature of a polygon point that is smooth (see the PolygonPoint.Smooth property for more information.) If a PolygonPoint is not smooth, the BezierControl has no effect.

Example The following example creates a PolygonGraphic and adds it to the diagram. One of it's PolygonPoint object's is made smooth, and then the BezierControl object is used to modify the curvature.

```
' Dimension the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
' Build a Polygon
With igxBuilder
    .BeginPath
    .MoveTo 0.3, 1
    .LineTo 0, 0.5
    .LineTo 0.5, 0
    .LineTo 1, 0.5
    .LineTo 0.7, 1
    .EndPath
    .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
   (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
' Smooth one of the PolygonPoints
igxGraphicObject.Graphic.PolygonGraphic.PolygonPoints.Item(3).Smooth
' Get the BezierControl from the PolygonPoint
Set igxBezier = igxGraphicObject.Graphic.PolygonGraphic _
   .PolygonPoints.Item(3).BezierControl
' Modify the Bezier handles
With igxBezier
    MsgBox "Move the first Bezier handle"
    .X1 = 0.2
    .Y1 = 0
    MsgBox "Move the second Bezier handle"
    .X2 = 0.8
    .Y2 = 0
End With
' Pause
MsgBox "Click OK to continue"
```

See Also BezierControl object

iGrafx API Object Hierarchy

{button PolygonPoint object,JI(`igrafxrf.HLP',`PolygonPoint_Object')}

Set Method

Topic Under Construction!!!

Syntax *PolygonPoint*.**Set**(*X* As Double, *Y* As Double)

Description The Set method sets the position of the specified PolygonPoint object within the coordinate space of a PolygonGraphic object. The *X* and *Y* arguments specify the coordinates for the point.

This method seems redundant with the X and Y Properties. Haven't yet investigated this. ???

Example The following example creates a polygon with five PolygonPoints. It then uses the Set method to move one of the points to a new position.

```
' Dimension the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
Dim igxPolyPoint As PolygonPoint
' Build a Polygon
With igxBuilder
   .BeginPath
   .MoveTo 0.3, 1
   .LineTo 0, 0.5
   .LineTo 0.5, 0
   .LineTo 1, 0.5
   .LineTo 0.7, 1
   .EndPath
   .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
   (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
MsgBox "Click OK to set a point to a new position"
Set igxPolyPoint = igxGraphicObject.Graphic.PolygonGraphic _
   .PolygonPoints.Item(3)
igxPolyPoint.Set 0.8, 0
MsgBox "Click OK to continue"
```

{button PolygonPoint object,JI(`igrafxrf.HLP',`PolygonPoint_Object')}

Smooth Method

Syntax *PolygonPoint*.**Smooth**

Description The Smooth method converts the specified PolygonPoint object to a smooth point. After converting the point to a smooth point, it's curve is a condensed curve, which looks just like a hard angle. Howver, the smooth point now has a BezierControl which can be modified to give the line segment a curved appearance as it passes through the point.

A “smoothed” PolygonPoint contributes to a curved line segment in a PolygonGraphic. A smooth PolygonPoint has a BezierControl that determines the curvature of the line segment as it passes through the point.

By contrast, an “unsmooth” PolygonPoint causes the PolygonGraphic's line segment to meet the point with hard angles; that is, with no curvature.

Example The following example creates a PolygonGraphic and adds it to the diagram. One of it's PolygonPoint object’s is made smooth, and then the BezierControl object is used to modify the curvature.

```
' Dimenson the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
' Build a Polygon
With igxBuilder
    .BeginPath
    .MoveTo 0.3, 1
    .LineTo 0, 0.5
    .LineTo 0.5, 0
    .LineTo 1, 0.5
    .LineTo 0.7, 1
    .EndPath
    .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
   (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
' Smooth one of the PolygonPoints
igxGraphicObject.Graphic.PolygonGraphic.PolygonPoints.Item(3).Smooth
' Get the BezierControl from the PolygonPoint
Set igxBezier = igxGraphicObject.Graphic.PolygonGraphic _
   .PolygonPoints.Item(3).BezierControl
' Modify the Bezier handles
With igxBezier
    MsgBox "Move the first Bezier handle"
    .X1 = 0.2
    .Y1 = 0
    MsgBox "Move the second Bezier handle"
    .X2 = 0.8
    .Y2 = 0
End With
' Pause
MsgBox "Click OK to continue"
```

See Also

BezierControl property

Unsmooth method

{button PolygonPoint object,JI(`igrafxrf.HLP',`PolygonPoint_Object')}

Unsmooth Method

Syntax *PolygonPoint*.**Unsmooth**

Description The Unsmooth method converts a smooth point to an unsmooth point. An unsmooth point causes the PolygonGraphic's line segment to meet the point with hard angles; that is, no curvature.

By contrast, a smooth PolygonPoint contributes to a curved line segment in the PolygonGraphic. A smooth PolygonPoint has a BezierControl that determines the curvature of the line segment as it passes through the point.

Example The following example creates a PolygonGraphic with one smooth PolygonPoint object. Then the Unsmooth method is used to unsmooth the point.

```
' Dimenson the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
Dim igxPolyPoint As PolygonPoint
' Build a Polygon
With igxBuilder
    .BeginPath
    .MoveTo 0.3, 1
    .LineTo 0, 0.5
    .LineTo 0.5, 0
    .LineTo 1, 0.5
    .LineTo 0.7, 1
    .EndPath
    .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
   (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
' Smooth one of the PolygonPoint
Set igxPolyPoint = igxGraphicObject.Graphic.PolygonGraphic _
   .PolygonPoints.Item(3)
igxPolyPoint.Smooth
' Get the BezierControl from the PolygonPoint
Set igxBezier = igxPolyPoint.BezierControl
' Modify the Bezier handles
With igxBezier
    .X1 = 0.2
    .Y1 = 0
    .X2 = 0.8
    .Y2 = 0
End With
' Unsmooth the PolygonPoint
MsgBox "The Polygon contains one Smooth PolygonPoint." _
   & Chr(13) & "Click OK to Unsmooth it."
igxPolyPoint.Unsmooth
' Pause
MsgBox "Click OK to continue."
```

See Also BezierControl property
Smooth method

{button PolygonPoint object,JI(`igrafxrf.HLP',`PolygonPoint_Object')}

X Property

Syntax *PolygonPoint*.**X**

Data Type Double (read/write)

Description The X property, along with the Y property, determines the position of the PolygonPoint within the coordinate space of the PolygonGraphic. The values normally range from 0.0 to 1.0, with 0.0 being the left-most position within the PolygonGraphic's bounding rectangle, and 1.0 being the right-most position in the PolygonGraphic's bounding rectangle.

You can use values greater than 1.0, and negative numbers, to exceed the boundaries of the PolygonGraphic.

Example The following example creates a PolygonGraphic with five PolygonPoints. Then one of the PolygonPoints is moved using the X and Y properties.

```
' Dimension the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
Dim igxPolyPoint As PolygonPoint
' Build a Polygon
With igxBuilder
    .BeginPath
    .MoveTo 0.3, 1
    .LineTo 0, 0.5
    .LineTo 0.5, 0
    .LineTo 1, 0.5
    .LineTo 0.7, 1
    .EndPath
    .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
   (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
' Smooth one of the PolygonPoint
Set igxPolyPoint = igxGraphicObject.Graphic.PolygonGraphic _
   .PolygonPoints.Item(3)
MsgBox "Click OK to move the PolyPoint."
igxPolyPoint.X = 0.8
igxPolyPoint.Y = 0
' Pause
MsgBox "Click OK to continue."
```

{button PolygonPoint object,JI(`igrafxrf.HLP',`PolygonPoint_Object')}

Y Property

Syntax *PolygonPoint*.**Y**

Data Type Double (read/write)

Description The Y property, along with the X property, determines the position of the PolygonPoint within the coordinate space of the PolygonGraphic. The values normally range from 0.0 to 1.0, with 0.0 being the left-most position within the PolygonGraphic's bounding rectangle, and 1.0 being the right-most position in the PolygonGraphic's bounding rectangle.

You can use values greater than 1.0, and negative numbers, to exceed the boundaries of the PolygonGraphic.

Example Refer to the Example for the X property .

{button PolygonPoint object,JI(`igrafxrf.HLP',`PolygonPoint_Object')}

PolygonPoints Object

The PolygonPoints object is a collection of individual PolygonPoint objects. A PolygonPoints collection is only associated with and accessible from the PolygonGraphic object.

The PolygonPoints object provides the following functionality:

- The ability to access any PolygonPoint objects in the collection.
- The ability to determine how many PolygonPoint objects are in the collection.
- The ability to add a new PolygonPoint object to the collection.

Through this object, the programmer gains access to a specific polygon point of a specific Polygon object.

The order of the polygon points in the collection is important, in that the order defines the direction of the line that describes the shape. This is important if you want to adjust a graphic by using the bezier control handles. See the BezierControl object.

Properties, Methods, and Events

All of the properties, methods, and events for the PolygonPoints object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent	RotatePoints	
	ScalePoints	
	TranslatePoints	

Related Topics

BezierControl object

PolygonGraphic object

PolygonPoint object

Add Method

Syntax *PolygonPoints*.**Add (***X* As Double, *Y* As Double)

Description The Add method adds a new PolygonPoint to the PolygonPoints collection. The *X* and *Y* arguments determine the position of the new PolygonPoint within the coordinate space of the shape.

The new PolygonPoint is added at the end of the collection of PolygonPoint objects, regardless of its X and Y position.

Example The following example creates a polygon with five PolygonPoints. Then a new PolygonPoint is added to the polygon.

```
' Dimension the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
Dim igxPolyPoint As PolygonPoint
' Build a Polygon
With igxBuilder
    .BeginPath
    .MoveTo 0.3, 1
    .LineTo 0, 0.5
    .LineTo 0.5, 0
    .LineTo 1, 0.5
    .LineTo 0.7, 1
    .EndPath
    .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
   (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
' Add one new PolygonPoint to the PolygonPoints collection
MsgBox "Click OK to add a new PolygonPoint to the " _
   & "PolygonPoints collection."
igxGraphicObject.Graphic.PolygonGraphic.PolygonPoints.Add 0.5, 0.7
' Pause
MsgBox "Click OK to continue."
```

{button PolygonPoints object,JI(`igrafxrf.HLP',`PolygonPoints_Object')}

Item Method

Syntax *PolygonPoints*.**Item**(*Index* As Integer) As PolygonPoint

Description The Item method returns a PolygonPoint object from the PolygonPoints collection. The *Index* argument determines which object to return. Use the Count property to determine the valid range of values for the *Index* argument.

Example The following example creates a PolygonGraphic and adds it to the diagram. One of it's PolygonPoint objects is selected using the Item method and made Smooth, and then the Point object's BezierControl object is used to modify the curvature.

```
' Dimension the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
' Build a Polygon
With igxBuilder
    .BeginPath
    .MoveTo 0.3, 1
    .LineTo 0, 0.5
    .LineTo 0.5, 0
    .LineTo 1, 0.5
    .LineTo 0.7, 1
    .EndPath
    .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
   (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
' Smooth one of the PolygonPoints
igxGraphicObject.Graphic.PolygonGraphic.PolygonPoints.Item(3).Smooth
' Get the BezierControl from the PolygonPoint
Set igxBezier = igxGraphicObject.Graphic.PolygonGraphic _
   .PolygonPoints.Item(3).BezierControl
' Modify the Bezier handles
With igxBezier
    MsgBox "Move the first Bezier handle"
    .X1 = 0.2
    .Y1 = 0
    MsgBox "Move the second Bezier handle"
    .X2 = 0.8
    .Y2 = 0
End With
' Pause
MsgBox "Click OK to continue
```

{button PolygonPoints object,JI(`igrafxrf.HLP',`PolygonPoints_Object')}

RotatePoints Method

Syntax *PolygonPoints*.**RotatePoints**(*CenterX* As Double, *CenterY* As Double, *Angle* As Double)

Description The RotatePoints method rotates the entire PolygonPoints collection as a unit. This effectively rotates the PolygonGraphic without rotating it's parent DiagramObject.

The *CenterX* and *CenterY* arguments specify the center of rotation within the shape's coordinate space. The values normally range from 0.0 to 1.0. The range of values may be different if the coordinate space of the shape was intensionally modified.

The *Angle* property specifies the amount of rotation in degrees. The Angle is additive, and relative to the current position, not relative to degree zero. Therefore, every time an angle is applied, that angle is added to the previous angle. Because the angle is relative, there is no automatic way to reset the rotation of the PolygonPoints back to the zero degree orientation.

Example The following example builds a polygon, and then rotates its PolygonPoints in 18 degree increments.

```
' Dimension the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
Dim igxPolyPoints As PolygonPoints
' Build a Polygon
With igxBuilder
   .BeginPath
   .MoveTo 0.3, 1
   .LineTo 0, 0.5
   .LineTo 0.5, 0
   .LineTo 1, 0.5
   .LineTo 0.7, 1
   .EndPath
   .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
    (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
' Smooth one of the PolygonPoint
Set igxPolyPoints = igxGraphicObject.Graphic.PolygonGraphic _
    .PolygonPoints
' Rotate the points is 18 degree increments
MsgBox "Click OK to rotate the PolygonPoints."
For Repeat = 1 To 20
   MsgBox "Rotated 18 degrees"
   igxPolyPoints.RotatePoints 0.5, 0.5, 18
Next Repeat
' Pause
MsgBox "Click OK to continue."
```

{button PolygonPoints object,JI(`igrafxrf.HLP',`PolygonPoints_Object')}

ScalePoints Method

Syntax *PolygonPoints*.**ScalePoints**(*CenterX* As Double, *CenterY* As Double, *XScale* As Double, *YScale* As Double)

Description The ScalePoints method scales the entire PolygonPoints collection as a unit. The result is a scaling of the PolygonGraphic without changing the size of it's parent DiagramObject.

The *CenterX* and *CenterY* arguments specify the *scaling center* within the DiagramObject's coordinate space. The values usually range from 0.0 to 1.0. You can use negative values and values greater than 1.0 to exceed the boundaries of the DiagramObject. The points move away from or toward the position specified by the *CenterX* and *CenterY* properties by the amount specifgied by the *XScale* and *YScale* arguments.

The *XScale* and *YScale* arguments are multipliers that specify the amount of scaling to apply to the PolygonPoints collection. The ScalePoints method calculates scaling by taking the distance of each PolygonPoint from the *scaling center*, and multiplying the distance by the amount specified with the *XScale* and *YScale* arguments. Values between 0.0 and 1.0 shrink the arrangement of the points. Values greater than 1.0 expand the arrangement of the points. Negative values can be used also. Negative values effectively move each point a negative distance, causing the arrangement to flip (mirror image) as well as scale, in one or both directions.

Example The following example builds a polygon, and then scales it's PolygonPoints to shrink the polygon.

```
' Dimension the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
Dim igxPolyPoints As PolygonPoints
' Build a Polygon
With igxBuilder
    .BeginPath
    .MoveTo 0.3, 1
    .LineTo 0, 0.5
    .LineTo 0.5, 0
    .LineTo 1, 0.5
    .LineTo 0.7, 1
    .EndPath
    .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
   (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
' Smooth one of the PolygonPoint
Set igxPolyPoints = igxGraphicObject.Graphic.PolygonGraphic _
   .PolygonPoints
ActiveDiagram.Selection.Add igxGraphicObject.DiagramObject
' Scale the points
MsgBox "Click OK to scale the PolygonPoints to half size."
igxPolyPoints.ScalePoints 0.5, 0.5, 0.5, 0.5
MsgBox "Click OK to continue."
```

{button PolygonPoints object,JI(`igrafxrf.HLP',`PolygonPoints_Object')}

TranslatePoints Method

Syntax *PolygonPoints*.**TranslatePoints**(*XOffset* As Double, *YOffset* As Double)

Description The TranslatePoints method moves the entire PolygonPoints collection as a unit. The polygon points are moved within the DiagramObject's coordinate space. This method results in moving the position of the PolygonGraphic without moving it's parent DiagramObject.

The *XOffset* argument specifies how far to move the polygon points horizontally. The distance is based on the DiagramObject's coordinate space, where the distance of one DiagramObject width equals 1.0. Positive values move the PolygonPoints collection to the right, and negative values move it to the left.

The *YOffset* argument specifies how far to move the polygon points vertically. The distance is based on the DiagramObject's coordinate space, where the distance of one DiagramObject height equals 1.0. Positive values move the PolygonPoints collection down, and negative values move it up.

The Offset values are relative to the current position, and are additive. Each time an Offset is applied, the PolygonPoints collection is offset from its current position.

Example The following example builds a polygon and adds it to the diagram. Then the PolygonPoints collection is scaled smaller, and finally translated to the right side of the DiagramObject.

```
' Dimension the variables
Dim igxGraphicObject As TextGraphicObject
Dim igxBuilder As New GraphicBuilder
Dim igxBezier As BezierControl
Dim igxPolyPoints As PolygonPoints
' Build a Polygon
With igxBuilder
   .BeginPath
   .MoveTo 0.3, 1
   .LineTo 0, 0.5
   .LineTo 0.5, 0
   .LineTo 1, 0.5
   .LineTo 0.7, 1
   .EndPath
   .Graphic.PolygonGraphic.Closed = True
End With
' Add a graphic to the diagram using the Polygon we built
Set igxGraphicObject = ActiveDiagram.DiagramObjects.AddGraphic _
   (igxBuilder.Graphic, 2880, 2880, 2880, 2880)
' Smooth one of the PolygonPoint
Set igxPolyPoints = igxGraphicObject.Graphic.PolygonGraphic _
   .PolygonPoints
ActiveDiagram.Selection.Add igxGraphicObject.DiagramObject
' Scale the points
MsgBox "Click OK to scale the PolygonPoints to half size."
igxPolyPoints.ScalePoints 0.5, 0.5, 0.5, 0.5
' Translate the PolygonPoints
MsgBox "Click OK to move the PolygonPoints."
igxPolyPoints.TranslatePoints 0.4, 0
MsgBox "Click OK to continue"
```

ImageGraphic Object

The ImageGraphic object represents some form of a computer graphics image; for instance, a bitmap image, a vector drawing, etc. The image must be pasted into the diagram as a metafile. The ImageGraphic object can then be derived from the Shape's *Graphic.ImageGraphic*. The ImageGraphic object does not accept metafile graphics derived from iGrafx Professional shapes.

The ImageGraphic object is subordinate to the Graphic object; that is, images are always, and exclusively, associated with a Graphic object, not a Shape object.

The ImageGraphic object allows you to adjust the size of the graphic, and set a mask color for the graphic.

Properties, Methods, and Events

All of the properties, methods, and events for the ImageGraphic object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Height		
MaskColor		
Parent		
UseMaskColor		
Width		

Related Topics

MetafileGraphic object

MaskColor Property

Syntax *ImageGraphic*.**MaskColor**

Data Type OLE_COLOR (read/write)

Description The MaskColor property specifies a color to be masked; that is, made transparent. Anywhere this color appears in the graphic, the graphic becomes transparent, and any graphics underneath the transparent area become visible.

The OLE_COLOR data type is the standard system color data type. You can supply a Long value if known, use Visual Basic's *RGB( )* function to supply a color, or use Visual Basic's color constants, such as *vbBlue*, *vbRed*, etc. The OLE_COLOR data value returned by MaskColor is a data type of Long.

Example The following example gets an ImageGraphic object and sets a mask color.

This example requires at least one image pasted into the diagram as a metafile. You might try pasting a bitmap image that contains some known primary color. Then you can modify this code to set the correct value for the MaskColor property.

```
' Dimension the variables
Dim igxGraphic As TextGraphicObject
Dim igxImageGraphic As ImageGraphic
' Find the first ImageGraphic on the diagram
For Index = 1 To ActiveDiagram.DiagramObjects.Count
    With ActiveDiagram.DiagramObjects.Item(Index)
        If (.Type = ixObjectShape) & (.Shape.Graphic.Type _
        = ixGraphicMetafile) Then
            Set igxImageGraphic = .Shape.Graphic.ImageGraphic
        End If
    End With
Next Index
MsgBox "Click OK to use a transparent mask color"
' Set the MaskColor. Change the color to work with your bitmap
igxImageGraphic.MaskColor = vbRed
' Use the Mask Color
igxImageGraphic.UseMaskColor = True
' Pause
MsgBox "Click OK to continue"
```

See Also UseMaskColor property

{button ImageGraphic object,JI(`igrafxrf.HLP',`ImageGraphic_Object')JumpId(igrafxrf.HLP,PolygonPoints_Object)}

UseMaskColor Property

Syntax *ImageGraphic*.**UseMaskColor**[= {**True | False**}]

Data Type Boolean (read/write)

Description The UseMaskColor property specifies whether the graphic uses a transparent mask color. If set to True, the MaskColor property is used, and anywhere that color appears in the graphic becomes transparent. If set to False, the graphic appears entirely opaque.

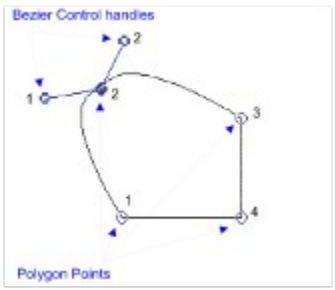
Example Refer to the example for the MaskColor property.

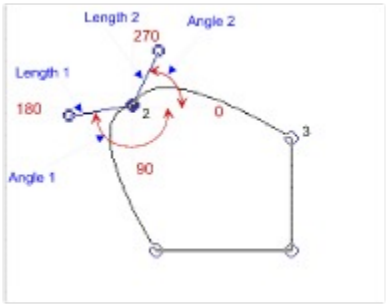
See Also MaskColor property

{button ImageGraphic object,JI(`igrafxrf.HLP',`ImageGraphic_Object')JumpId(igrafxrf.HLP,PolygonPoints_Object)}

BezierControl Object

The BezierControl object provides the means for manipulating or adjusting bezier spline curves, which are used extensively in drawing many types of graphical shapes. The following diagram illustrates the concept of a Bezier control, and provides the terminology to understand the use of this object.





There is a BezierControl object for every polygon point that makes up a graphic. The left side of the illustration shows the numbered polygon points that describe the shape, and two Bezier Control handles protruding from the second polygon point. Control handle 1 affects the shape of the line segment between polygon points 1 and 2. Control handle 2 affects the shape of the line segment between polygon points 2 and 3. However, as you may be able to see, the line segments affected more strongly at the ends of the line segments that are closest to polygon point displaying the control handles.

The right side of the illustration shows some relationships that are important for programming with the BezierControl object. The red line on polygon point 2 shows the coordinate system for specifying a value for the Angle1 and Angle2 properties. This coordinate system layout is the same for all polygon points, regardless of their orientation.

The following list describes the various properties you can use to position the bezier control handles, and how moving the handle affects the associated line segment.

- X and Y position
- X Offset and Y Offset
- Length
- Angle

Example

The following example is referenced for every property of this object. The example sets up a Shape with a PolygonGraphic and a Bezier Control handle. Also on the diagram is a TextGraphicObject that displays the current values of all the BezierControl object's properties. The Shape has two Adjustment Points, one for each BezierControl handle. When a handle is moved, the on-screen information is updated. Use this example to:

- Determine how the properties are related
- Determine how each property relates to the position of a Bezier handle
- Determine useful values for each of the properties

```
' Dimension module variables
Private WithEvents igxShape As Shape
Private igxInfo As TextGraphicObject
Private igxBezier As BezierControl

Private Sub Main()
   ' Dimension the variables
   Dim igxBuilder As New GraphicBuilder
   Dim igxPolygon As PolygonGraphic
```

```
   ' Build a Polygon
   With igxBuilder
      .BeginPath
      .MoveTo 0.3, 1
      .LineTo 0, 0.5
      .LineTo 0.5, 0
      .LineTo 1, 0.5
      .LineTo 0.7, 1
      .EndPath
      .Graphic.PolygonGraphic.Closed = True
   End With
   ' Add a graphic to the diagram using the Polygon we built
   Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
   igxShape.Graphic.Replace igxBuilder.Graphic
   igxShape.DiagramObject.Width = 1440 * 2
   igxShape.DiagramObject.Height = 1440 * 2
   Set igxInfo = ActiveDiagram.DiagramObjects.AddTextObject _
     (1440 * 5, 1440 * 4)
   igxInfo.DiagramObject.Width = 1440 * 2
   igxInfo.DiagramObject.Height = 1440 * 5
   ' Get the Shape's Polygon object
   Set igxPolygon = igxShape.Graphic.PolygonGraphic
   ' Smooth one of the PolygonPoints
   igxPolygon.PolygonPoints.Item(3).Smooth
   ' Get the BezierControl from the PolygonPoint
   Set igxBezier = igxPolygon.PolygonPoints.Item(3).BezierControl
   ' Initial Bezier adjustments
   igxBezier.X1 = 0.2
   igxBezier.Y1 = 0
   igxBezier.X2 = 0.8
   igxBezier.Y2 = 0
   ' Add two adjustment points to the shape
   igxShape.Adjustments.Add igxBezier.X1, igxBezier.Y1
   igxShape.Adjustments.Add igxBezier.X2, igxBezier.Y2
   ' Select the shape
   ActiveDiagram.Selection.Add igxShape.DiagramObject
   ' Update our on-screen info
   UpdateInfo
   ' Pause
   MsgBox "Return to the diagram and move the adjustment handles."
End Sub

' When an adjustment point moves update the Bezier handles and info
Private Sub igxShape_AdjustmentMove(ByVal Index As Integer, X As Double, Y As
Double)
   If Index = 1 Then
      igxBezier.X1 = X
      igxBezier.Y1 = Y
      UpdateInfo
   Else
```

```
      igxBezier.X2 = X
      igxBezier.Y2 = Y
      UpdateInfo
   End If
   Dim sString1 As String
   Dim sString2 As String
End Sub

' Update the info on the text object
Private Sub UpdateInfo()
   Dim sString1 As String
   Dim sString2 As String
   ' Gather Handle 1 info
   sString1 = _
       "Angle: " & igxBezier.Angle1 & Chr(13) & _
       "Length: " & igxBezier.Length1 & Chr(13) & _
       "X: " & igxBezier.X1 & Chr(13) & _
       "Y: " & igxBezier.Y1 & Chr(13) & _
       "XOffset: " & igxBezier.X1Offset & Chr(13) & _
       "YOffset: " & igxBezier.Y1Offset & Chr(13)
   ' Gather Handle 2 info
   sString2 = _
       "Angle: " & igxBezier.Angle2 & Chr(13) & _
       "Length: " & igxBezier.Length2 & Chr(13) & _
       "X: " & igxBezier.X2 & Chr(13) & _
       "Y: " & igxBezier.Y2 & Chr(13) & _
       "XOffset: " & igxBezier.X2Offset & Chr(13) & _
       "YOffset: " & igxBezier.Y2Offset & Chr(13)
   ' Change the shape text
   igxInfo.Text = "Bezier Handle 1:" & Chr(13) & Chr(13) & _
     sString1 & Chr(13) & Chr(13) & _
     "Bezier Handle 2:" & Chr(13) & Chr(13) & sString2
End Sub
```

Properties, Methods, and Events

All of the properties, methods, and events for the BezierControl object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Angle1		
Angle2		
Application		
Length1		
Length2		
Parent		
X1		
X1Offset		
X2		

X2Offset
Y1
Y1Offset
Y2
Y2Offset

Related Topics

PolygonPoint object

Angle1 Property

Syntax *BezierControl*.**Angle1**

Data Type Double (read/write)

Description The Angle1 property specifies the angle, in degrees, for Control Handle Arm 1. The value specifies the number of degrees from horizontal. Valid values for this property are between 0 and +359.

Example Refer to the example provided in the BezierControl object topic.

See Also Angle2 property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

Angle2 Property

Syntax *BezierControl*.**Angle2**

Data Type Double (read/write)

Description The Angle2 property specifies the angle, in degrees, for Control Handle Arm 2. The value specifies the number of degrees from horizontal. Valid values for this property are between 0 and +359.

Example Refer to the example provided in the BezierControl object topic.

See Also Angle1 property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

Length1 Property

Syntax *BezierControl*.**Length1**

Data Type Double (read/write)

Description The Length1 property specifies the length of Control Handle Arm 1. The length is relative to the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0.

Example Refer to the example provided in the BezierControl object topic.

See Also Length2 property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

Length2 Property

Syntax *BezierControl*.**Length2**

Data Type Double (read/write)

Description The Length1 property specifies the length of Control Handle Arm 1. The length is relative to the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0.

Example Refer to the example provided in the BezierControl object topic.

See Also Length1 property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

X1 Property

Syntax *BezierControl*.**X1**

Data Type Double (read/write)

Description The X1 property specifies the position of Control Handle 1 in the X direction. The position value is based on the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0.

Example Refer to the example provided in the BezierControl object topic.

See Also X2 property

Y1 property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

X1Offset Property

Syntax *BezierControl*.**X1Offset**

Data Type Double (read/write)

Description The X1Offset property specifies a distance to move Control Handle 1 horizontally from its current position. The distance is based on the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0. Negative values move the Control Handle to the left, and positive values move the Control Handle to the right.

Example Refer to the example provided in the BezierControl object topic.

See Also X2Offset property

Y1Offset property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

X2 Property

Syntax *BezierControl*.**X2**

Data Type Double (read/write)

Description The X2 property specifies the position of Control Handle 2 in the X direction. The position value is based on the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0.

Example Refer to the example provided in the BezierControl object topic.

See Also X1 property

Y2 property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

X2Offset Property

Syntax *BezierControl*.**X2Offset**

Data Type Double (read/write)

Description The X2Offset property specifies a distance to move Control Handle 2 horizontally from its current position. The distance is based on the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0. Negative values move the Control Handle to the left, and positive values move the Control Handle to the right.

Example Refer to the example provided in the BezierControl object topic.

See Also X1Offset property

Y2Offset property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

Y1 Property

Syntax *BezierControl*.**Y1**

Data Type Double (read/write)

Description The Y1 property specifies the position of Control Handle 1 in the Y direction. The position value is based on the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0.

Example Refer to the example provided in the BezierControl object topic.

See Also Y2 property

X1 property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

Y1Offset Property

Syntax *BezierControl*.**Y1Offset**

Data Type Double (read/write)

Description The Y1Offset property specifies a distance to move Control Handle 1 vertically from its current position. The distance is based on the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0. Negative values move the Control Handle up, and positive values move the Control Handle down.

Example Refer to the example provided in the BezierControl object topic.

See Also Y2Offset property

X1Offset property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

Y2 Property

Syntax *BezierControl*.**Y2**

Data Type Double (read/write)

Description The Y2 property specifies the position of Control Handle 2 in the Y direction. The position value is based on the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0.

Example Refer to the example provided in the BezierControl object topic.

See Also X2 property

Y1 property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

Y2Offset Property

Syntax *BezierControl*.**Y2Offset**

Data Type Double (read/write)

Description The Y2Offset property specifies a distance to move Control Handle 2 vertically from its current position. The distance is based on the DiagramObject's coordinate system, and is typically a value between 0.0 and 1.0. Negative values move the Control Handle up, and positive values move the Control Handle down.

Example Refer to the example provided in the BezierControl object topic.

See Also Y1Offset property

X2Offset property

{button BezierControl object,JI(`igrafxrf.HLP',`BezierControl_Object')}

MetafileGraphic Object

The MetafileGraphic object represents a vector graphic. When you insert Microsoft Office Clipart into a diagram, it becomes a Shape object with a MetafileGraphic. MetafileGraphic objects can be made of multiple graphic elements, so they are similar to the GraphicGroup object, but must be converted before treated as a GraphicGroup. The MetafileGraphic object has one method, which converts a MetafileGraphic object into a GraphicGroup object.

Properties, Methods, and Events

All of the properties, methods, and events for the MetafileGraphic object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	ConvertToGroup	
Parent		

Related Topics

ImageGraphic object

ConvertToGroup Method

Syntax *MetafileGraphic*.**ConvertToGroup**

Description The ConvertToGroup method converts a MetafileGraphic object into a GraphicGroup object. This method as two effects:

- The MetafileGraphic is changed to a GraphicGroup. The Shape's Graphic property then returns a GraphicGroup object instead of a MetafileGraphic object. The Shape's Graphic.MetafileGraphic property is no longer valid.
- The Shape's Graphic.Type property changes from *ixGraphicMetafile* to *ixGraphicGroup*.

Because the ConvertToGroup method changes it's parent Graphic object, it must be used in the context of a Graphic object. If used in the context of a MetafileGraphic, the MetafileGraphic becomes invalid. For instance:

The following code statement converts the Graphic object correctly:

```
Shape.Graphic.MetafileGraphic.ConvertToGroup
```

The following code statement makes the MyMetafileGraphic variable invalid:

```
Set MyMetafileGraphic = Shape.Graphic.MetafileGraphic
MyMetafileGraphic.ConvertToGroup
```

Example The following example converts a MetafileGraphic to a GraphicGroup. It then uses the GraphicGroup count property to report the number of graphic elements in the graphic.

The example requires at least one Clipart image inserted into the diagram.

```
' Dimension the variables
Dim igxMetafileGraphic As MetafileGraphic
Dim igxShape As Shape
Dim igxGraphicGroup As GraphicGroup
' Find the first Shape with MetafileGraphic in the diagram
With ActiveDiagram.DiagramObjects
   For Index = 1 To .Count
      If (.Item(Index).Type = ixObjectShape) Then
         If .Item(Index).Shape.Graphic.Type = _
         ixGraphicMetafile Then
             ' If found, get the Shape Object
             Set igxShape = .Item(Index).Shape
             Exit For
         End If
      End If
   Next Index
End With
igxShape.Graphic.MetafileGraphic.ConvertToGroup
Set igxGraphicGroup = igxShape.Graphic.GraphicGroup
MsgBox "The converted group has " & igxGraphicGroup.Graphics.Count _
    & " graphic elements."
```

{button MetafileGraphic object,JI(`igrafxrf.HLP',`MetafileGraphic_Object')}

GraphicBuilder Object

The GraphicBuilder object provides functionality for drawing graphical symbols using a variety of tools. Graphical objects created using the GraphicBuilder can be made into Graphic objects, TextGraphicObject objects, or Shape objects. Any of the valid graphic primitive types can be created or output from the GraphicBuilder.

To use the GraphicBuilder object in your code, you need the following line of setup code:

```
Dim igxGraphicBuilder As New GraphicBuilder
```

Properties, Methods, and Events

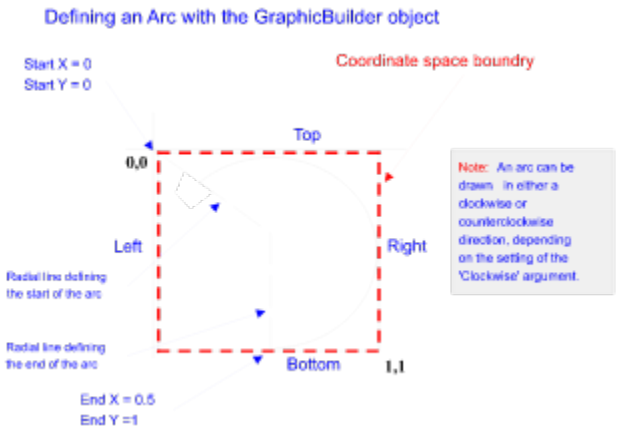
All of the properties, methods, and events for the GraphicBuilder object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Arc	
Graphic	BeginClipPath	
Height	BeginPath	
Parent	BeginSpline	
Width	BezierTo	
	BSplineTo	
	Chord	
	Circle	
	CircularArcTo	
	Close	
	Ellipse	
	Ellipse2	
	EllipticalArcTo	
	EndClipPath	
	EndPath	
	LineTo	
	MoveTo	
	Pie	
	Polygon	
	Rectangle	
	Reset	
	SplineKnot	
	Star	

Arc Method

Syntax *GraphicBuilder*.**Arc**(*Left* As Double, *Top* As Double, *Right* As Double, *Bottom* As Double, *XStart* As Double, *YStart* As Double, *XEnd* As Double, *YEnd* As Double, *Clockwise* As Boolean)

Description The Arc method draws an elliptical arc. The resulting graphic is an ArcGraphic object, with the ArcType property set to ixArcNormal. Arcs are drawn based on defining two radial lines (a "start" line and an "end" line) that extend from the center of the coordinate space. The following illustration depicts how an arc is drawn, and the meaning of the arguments.



The arguments for this method are as follows:

The *Left* argument specifies the position along the X coordinate axis of the upper left corner of the bounding rectangle of the arc.

The *Top* argument specifies the position along the Y coordinate axis of the upper left corner of the bounding rectangle of the arc.

The *Right* argument specifies the position along the X coordinate axis of the lower right corner of the bounding rectangle of the arc.

The *Bottom* argument specifies the position along the Y coordinate axis of the lower right corner of the bounding rectangle of the arc.

The *XStart* argument specifies the position along the X coordinate axis of the end point of a radial line that extends from the center of the coordinate space. It is this radial line that defines the starting point of the arc.

The *YStart* argument specifies the position along the Y coordinate axis of the end point of a radial line that extends from the center of the coordinate space. It is this radial line that defines the starting point of the arc.

The *XEnd* argument specifies the position along the X coordinate axis of the end point of a radial line that extends from the center of the coordinate space. It is this radial line that defines the ending point of the arc.

The *YEnd* argument specifies the position along the X coordinate axis of the end point of a radial line that extends from the center of the coordinate space. It is this radial line that defines the ending point of the arc.

The points defined by the Left, Top, Right, and Bottom arguments specify the bounding rectangle of the arc. An ellipse formed by the bounding rectangle defines the curve of the arc. The arc extends in the current drawing direction from the point where it intersects the radial from the center of the bounding rectangle to the point defined by the *XStart* and *YStart* arguments. The arc ends where it intersects the radial from the center of the bounding rectangle to the point defined by the *XEnd* and *YEnd* arguments. If the starting and ending points are the same, a complete (closed) ellipse is drawn.

If you are drawing an arc inside a path, the Arc method continues the path by drawing a line from the current position to the beginning of the arc, resulting in a polygon. The current position is updated to be the end of the arc. The arc is drawn using the current pen, and is not filled.

Example The following example places two shapes in the active diagram. The Arc method is then used to draw a graphic with the GraphicBuilder. The Arc graphic is used to replace the graphic in the first shape. The GraphicBuilder is then reset, and another Arc is drawn, which is then used to replace Shape 2's graphic.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create two shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "Shapes added to diagram. Replace the shape graphics" _
    & Chr(13) & "with new graphics from the GraphicBuilder."
' Create an Arc graphic with the GraphicBuilder
igxGrfxBuilder.Arc 0.3, 0, 0.8, 0.9, 1, 0.6, 0.4, 0, False
' Replace the graphic inside the first shape with the new graphic
igxShape1.Graphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
' Reset the GraphicBuilder and draw a new shape
igxGrfxBuilder.Reset
' Create another Arc graphic with the GraphicBuilder
igxGrfxBuilder.Pie 0.3, 0.2, 0.8, 0.7, 1, 0.5, 0.3, 0, True
' Replace the graphic inside the second shape with the new graphic
igxShape2.Graphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
```

See Also ArcGraphic object

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

BSplineTo Method

Syntax *GraphicBuilder*.**BSplineTo**(*X* As Double, *Y* As Double)

Description The BSplineTo method draws a B-Spline to the coordinate point specified by the *X* and *Y* arguments. The values of the *X* and *Y* arguments represent a point in the relative coordinate space of the graphic, which typically ranges from 0.0 to 1.0.

Example The following example uses the BSplineTo method to create a new graphic for a shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "Shape added to diagram. Replace the shape's graphic" _
    & Chr(13) & "with a new graphic from the GraphicBuilder."
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create the graphic
igxGrfxBuilder.BeginPath
' Move the pen to 0,0.5 in the coordinate space
igxGrfxBuilder.MoveTo 0, 0.5
' Draw a new graphical shape with BSplines and line segments
igxGrfxBuilder.BSplineTo 0.5, 0
igxGrfxBuilder.LineTo 1, 0
igxGrfxBuilder.BSplineTo 0.5, 0.5
igxGrfxBuilder.BSplineTo 1, 1
igxGrfxBuilder.LineTo 0.5, 1
igxGrfxBuilder.Close
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
```

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

BeginClipPath Method

Syntax *GraphicBuilder*.**BeginClipPath**

Description The BeginClipPath method specifies that all subsequent objects, until the EndClipPath method is called, are part of a “clipping path”. A clipping path defines a closed graphic that is used in a Boolean Intersection operation.

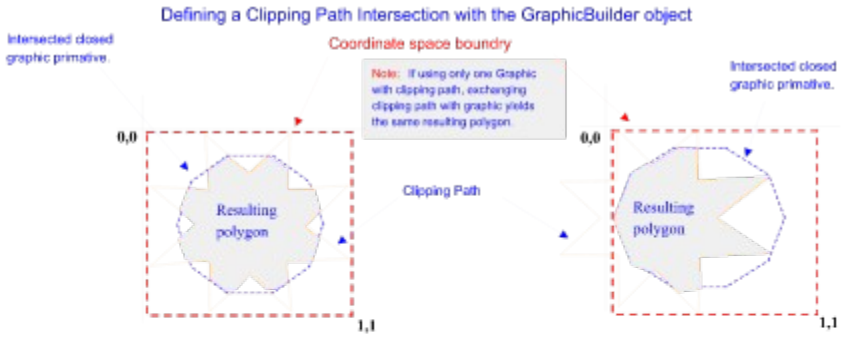
A clipping path can be any closed graphic primative; for example, an ellipse, polygon, circle, rectangle, pie, etc. The process of creating a clipping path and a graphic are as follows:

- Call the BeginClipPath method
- Draw a closed graphic, using any method or method you want.
- Call the EndClipPath method (this is your clipping area)
- Draw a graphic to intersect with the clipping path. The resulting graphic is the **Intersection** of the two or more closed graphic primatives (not a Boolean Subraction).

When a clipping path is created, it is not visible in the diagram. Only when other polygons intersect it is it visible where the intersecton takes place.

If more than one closed graphic primative is used with one clipping path, the first resulting graphic is drawn, then the next resulting graphic is drawn over the first graphic (see illustration).

The following illustration shows how a clipping path is used.



A Star and Polygon Intersection made with a clipping path.

The results of multiple closed graphic primatives used on a clipping path.

Example The following example creates a graphic by defining an eight-point star as the clipping path for a ten-sided polygon.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
```

```
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin the clip path
igxGrfxBuilder.BeginClipPath
' Define an 8 point star
igxGrfxBuilder.Star 0.5, 0.5, 0.25, 0.5, 8, 0
' End the clip path
igxGrfxBuilder.EndClipPath
' Create a 10 sided polygon to intersect with the Star
igxGrfxBuilder.Polygon 0.5, 0.5, 0.35, 10, 0
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
```

See Also EndClipPath method

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

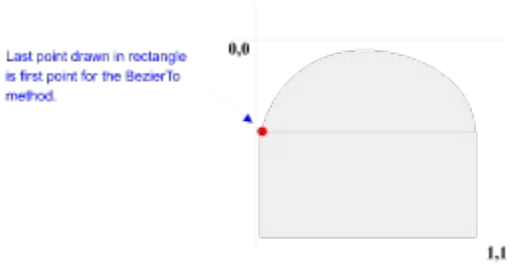
BeginPath Method

Syntax *GraphicBuilder*.**BeginPath**

Description The BeginPath method is used in conjunction with the EndPath method to draw polygons with the GraphicBuilder object, using a point-by-point method. Anything you can draw with a mouse may be done with this approach. You can use any valid GraphicBuilder method to draw inside a BeginPath/EndPath bracket. The resulting Graphic “type” can vary depending on what you draw, and how you draw it.

If a closed graphic primative such as a rectangle is defined, it is drawn normally but may be modified by the LineTo, BSplineTo, BezierTo,CircularArcTo or EllipticalArcTo methods. Since the rectangle is drawn from points 1 to 4, once it ends on point 4 another line may be drawn from that location that extends the original primative to become part of a more complex graphic (see illustration).

The following illustration shows how you can use a rectangle and a spline to draw more complicated graphical symbols. The graphic shown in the illustration (and drawn using the example code) is considered a PolyPolygonGraphic



Example The following example creates the graphic shown in the illustration.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape on the active diagram.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create the polygon
igxGrfxBuilder.BeginPath
' Build the rectangle
igxGrfxBuilder.Rectangle 0, 0.5, 1, 0.5, 0
' Continue the graphic with a bezier curve
igxGrfxBuilder.BezierTo 0.2, 0, 0.8, 0, 1, 0.5
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to gray
igxShape.FillColor = RGB(240, 240, 240)
MsgBox "The shape's graphic type is " & igxShape.Graphic.Type
```

See Also EndPath method

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

BezierTo Method

Syntax *GraphicBuilder*.**BezierTo**(*X* As Double, *Y* As Double, *X1* As Double, *Y1* As Double, *X2* As Double, *Y2* As Double)

Description The BezierTo method creates a segment of a jointed Bezier curve. Two control handles are used to change the shape of the curve. The BezierTo method is used within the brackets of the BeginPath and EndPath methods.

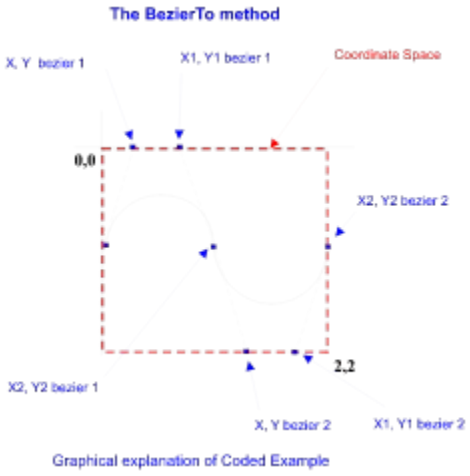
The *X* and *Y* arguments define the coordinates for the control handle of the first point (the location of the first point is defined by the previous MoveTo method).

The *X1* and *Y1* arguments define the coordinates for the control handle of the second point.

The *X2* and *Y2* arguments define the coordinates for the second point of the curve.

All coordinates are in relation to the graphic's coordinate space. A Bezier curve's handles only affect itself, and do not change adjoining lines or curves.

The following illustration shows how to use the BezierTo method to construct a line segment.



Example The following example draws the "sine wave" line shown in the illustration.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create the graphic
igxGrfxBuilder.BeginPath
' Move the pen to 0,0.5 in the coordinate space
igxGrfxBuilder.MoveTo 0, 0.5
' Draw a sine wave
igxGrfxBuilder.BezierTo 0.2, 0, 0.8, 0, 1, 0.5
igxGrfxBuilder.BezierTo 1.2, 1, 1.8, 1, 2, 0.5
igxGrfxBuilder.EndPath
```

```
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
```

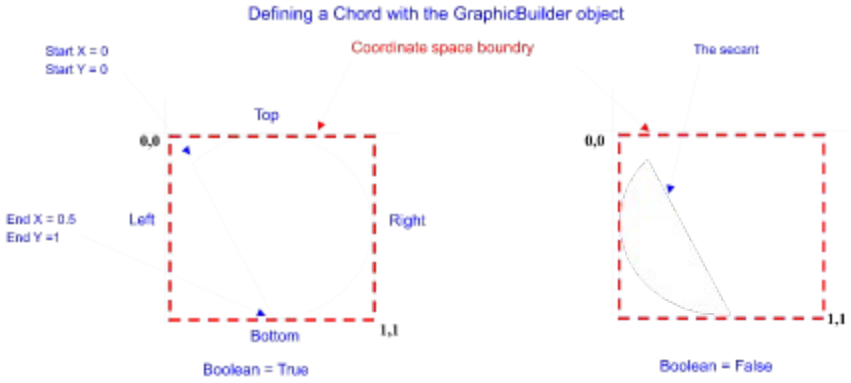
{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Chord Method

Syntax *GraphicBuilder*.**Chord**(*Left* As Double, *Top* As Double, *Right* As Double, *Bottom* As Double, *StartX* As Double, *StartY* As Double, *EndX* As Double, *EndY* As Double, *Clockwise* As Boolean)

Description The Chord method draws a chord, which is a region bounded by the intersection of an ellipse and a line segment (called a “secant”). The resulting graphic is an ArcGraphic object, with the ArcType property set to ixArcChord. The chord is outlined using the current pen, and is filled using the current brush. The current position is neither used nor updated by the Chord method.

The following illustration shows the definintion of a chord and its properties.



The arguments for this method are as follows:

The *Left* argument specifies the position along the X coordinate axis of the upper left corner of the bounding rectangle of the chord.

The *Top* argument specifies the position along the Y coordinate axis of the upper left corner of the bounding rectangle of the chord.

The *Right* argument specifies the position along the X coordinate axis of the lower right corner of the bounding rectangle of the chord.

The *Bottom* argument specifies the position along the Y coordinate axis of the lower right corner of the bounding rectangle of the chord.

The *XRadial1* argument specifies the position along the X coordinate axis of the end point of the radial line that defines the beginning of the chord.

The *YRadial1* argument specifies the position along the Y coordinate axis of the end point of the radial line that defines the beginning of the chord.

The *XRadial2* argument specifies the position along the X coordinate axis of the end point of the radial line that defines the end of the chord.

The *YRadial2* argument specifies the position along the X coordinate axis of the end point of the radial line that defines the end of the chord.

The points defined by the Left, Top, Right, and Bottom arguments specify the bounding rectangle of the chord. An ellipse formed by the bounding rectangle defines the curve of the chord. The curve begins at the point where the ellipse intersects the first radial, and extends counterclockwise to the point where the ellipse intersects the second radial. (A radial is a line segment drawn from the center of an ellipse to a specified endpoint on an ellipse.) The chord is closed by drawing a line from the intersection of the first radial and the curve, to the intersection of the second radial and the curve. If the starting and ending points are the same, a complete (closed) ellipse is drawn.

Example The following example creates two parts of an ellipse and offsets one from the other using the chord method.

```
' Dimension the variables
```

```
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Define the first chord
igxGrfxBuilder.Chord 0, 0, 1, 1, 0, 0, 1, 1, True
' Define the other half with a false boolean and offset it.
igxGrfxBuilder.Chord 0, 0, 1, 1, 0, 0, 0.9, 1.1, False
' Replace the graphic inside the shape with the new graphic.
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to blue.
igxShape.FillColor = vbBlue
```

See Also

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Circle Method

Topic Under Construction!!! Code needs to be verified.

Syntax *GraphicBuilder*.**Circle**(*CenterX* As Double, *CenterY* As Double, *Radius* As Double)

Description The Circle method draws a circle as defined by a center position and a radius.

The *CenterX* and *CenterY* arguments define the center of the circle within the graphic's coordinate space, usually a value between 0.0 and 1.0.

The *Radius* argument defines the size of the circle as a radius from the center point. This argument also is defined in terms of the graphic's coordinate space, usually a value between 0.0 and 1.0.

Example The following example adds a shape in the diagram, and then draws a circle using the GraphicBuilder's Circle method. The shape's graphic is replaced with the circle. Then another circle is drawn, making the GraphicBuilder's Graphic a GraphicGroup consisting of two circles. The shape's graphic is then replaced with the group.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "Shape added to diagram. Replace the shape's graphic" _
    & Chr(13) & "with a new graphic from the GraphicBuilder."
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Create a circle
igxGrfxBuilder.Circle 0.25, 0.25, 0.25
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
' Draw another circle
igxGrfxBuilder.Circle 0.75, 0.75, 0.25
' The GraphicBuilder's Graphic is now a Group
' Assign the Group to the shape
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
```

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

CircularArcTo Method

Syntax *GraphicBuilder*.**CircularArcTo**(*X* As Double, *Y* As Double, *Bow* As Double)

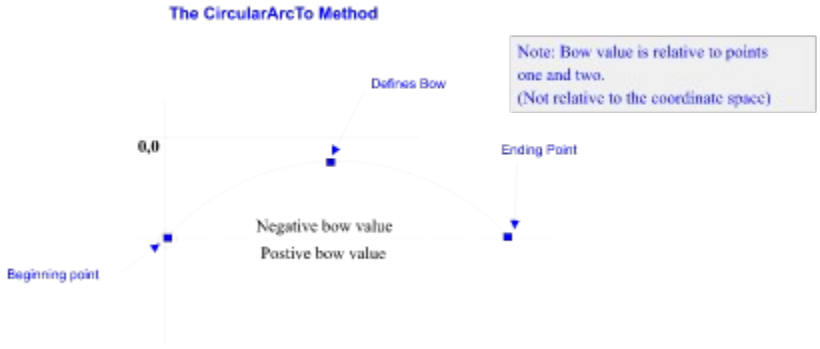
Description The CircularArcTo method draws a circular arc, and must be used within the BeginPath and EndPath statements. The CircularArcTo method continues the path by drawing a circular arc from the current position to the location specified by the method's *X* and *Y* arguments. The current position is then updated to be the end of the arc. The arc is drawn using the current pen, and is not filled. The resulting Graphic "type" can vary depending on what you draw, and how you draw it.

The *X* argument specifies the X coordinate position of the end point of the arc.

The *Y* argument specifies the Y coordinate position of the end point of the arc.

The *Bow* argument specifies the amount and direction of bow for the circular arc.

The following illustration shows a circular arc, and how the *Bow* argument controls the shape of the arc.



Example The following example shows how to use the CircularArcTo method.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create the graphic
igxGrfxBuilder.BeginPath
' Move the pen to 0,0.5 in the coordinate space
igxGrfxBuilder.MoveTo 0, 0.5
' Draw a circular arc
igxGrfxBuilder.CircularArcTo 1, 0.5, -0.2
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
```

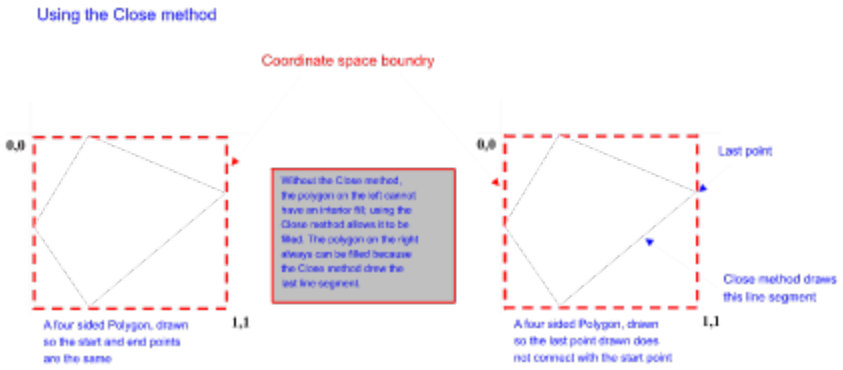
{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Close Method

Syntax *GraphicBuilder*.**Close**

Description The Close method creates a closed polygon. Closed polygons can have an interior fill; open polygons cannot. In drawing a polygon, you can either connect the starting point with the end point yourself using any of the available line drawing methods, or you can use the close method to draw the last line segment as a straight line.

The following illustration shows the two ways the Close method can be used when creating polygons or other graphics drawn with line segments.



Example The following example illustrates how to use the Close method. First, a shape is created in the active diagram. Then the GraphicBuilder is used to draw two polygons. The first polygon is drawn so it connects itself at the starting point, and the Close method is called to close it so it can be filled. The second polygon is also drawn so it connects at its starting point, but the Close method is not called. The shape's graphic is replaced, and then the shape's fill color is set to green. The first polygon is filled but the second one is not. Next, another shape is added to the diagram, and another polygon is drawn with the GraphicBuilder. This time, the Close method is used to complete the polygon (a straight line segment is drawn from the last point created with the LineTo method to the starting point).

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create first polygon
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 0.1, 1
igxGrfxBuilder.LineTo 0.1, 0.3
igxGrfxBuilder.LineTo 0.9, 0.3
```

```
igxGrfxBuilder.LineTo 0.9, 1
igxGrfxBuilder.LineTo 0.1, 1
' Close the polygon so it can be filled
igxGrfxBuilder.Close
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Create path for other polygon
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 0.1, 0.2
igxGrfxBuilder.LineTo 0.1, 0
igxGrfxBuilder.LineTo 0.9, 0
igxGrfxBuilder.LineTo 0.9, 0.2
igxGrfxBuilder.LineTo 0.1, 0.2
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to green
igxShape.FillColor = vbGreen
MsgBox "View the diagram"
' Create a second shape
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Start fresh with the GraphicBuilder
igxGrfxBuilder.Reset
' Begin a path to create first polygon
igxGrfxBuilder.BeginPath
' Build a polygon and use the Close method to draw the last line
igxGrfxBuilder.MoveTo 0.1, 1
igxGrfxBuilder.LineTo 0, 0.3
igxGrfxBuilder.LineTo 0.5, 0
igxGrfxBuilder.LineTo 1, 0.5
' Close the polygon
igxGrfxBuilder.Close
' End the path to make the polygon
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to blue
igxShape.FillColor = vbBlue
MsgBox "View the diagram"
```

Ellipse Method

Syntax *GraphicBuilder*.**Ellipse**(*Left* As Double, *Top* As Double, *Width* As Double, *Height* As Double)

Description The Ellipse method is used to create an EllipseGraphic object.

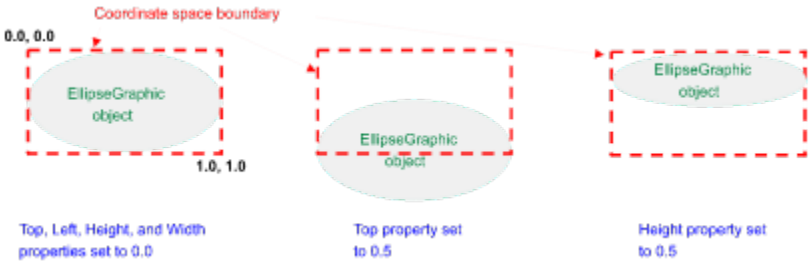
The *Left* argument is a value from 0.0 to 1.0 that specifies the position of the left side of the ellipse graphic.

The *Top* argument is a value from 0.0 to 1.0 that specifies the position of the top of the ellipse graphic.

The *Width* argument is a value from 0.0 to 1.0 that specifies the width of the ellipse graphic.

The *Height* argument is a value from 0.0 to 1.0 that specifies the height of the ellipse graphic.

The following illustration shows how setting the argument values affect the construction of the ellipse.



Example The following example creates a shape on the active diagram, and then replaces the shape's graphic with an ellipse created using the GraphicsBuilder object.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add an ellipse to the graphic
igxGrfxBuilder.Ellipse 0.1, 0.1, 0.8, 0.8
' Display a message box before replacing shape
MsgBox "Click OK to replace shape with an ellipse."
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
```

See Also Ellipse2 method

EllipseGraphic object

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Ellipse2 Method

Syntax *GraphicBuilder*.**Ellipse2**(*CenterX* As Double, *CenterY* As Double, *RadiusX* As Double, *RadiusY* As Double)

Description The Ellipse2 method is used to create an EllipseGraphic object. It is alternative method, compared to the Ellipse method, that defines the ellipse based on a center position and a radius in both the X and Y coordinates.

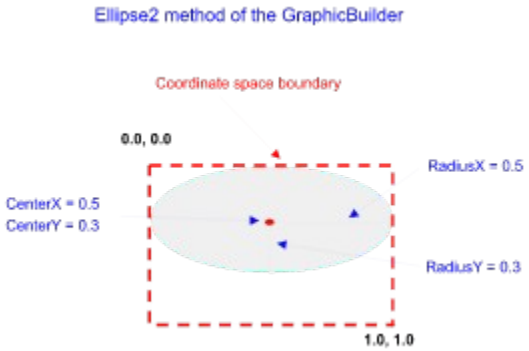
The *CenterX* argument defines the center of the ellipse in the X direction of the coordinate space. The value is typically between 0.0 and 1.0.

The *CenterY* argument defines the center of the ellipse in the Y direction of the coordinate space. The value is typically between 0.0 and 1.0.

The *RadiusX* argument defines the radius of the ellipse in the X direction from the center point.

The *RadiusY* argument defines the radius of the ellipse in the Y direction from the center point.

The following illustration shows how the arguments define the ellipse.



Example The following example creates a shapes in the active diagram. It then creates an ellipse with the GraphicBuilder's Ellipse2 method. The shape's graphic is then replaced with the new ellipse.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add an ellipse to the graphic using the Ellipse2 method
igxGrfxBuilder.Ellipse2 0.5, 0.3, 0.5, 0.3, 45
' Display a message box before replacing shape
MsgBox "Click OK to replace shape with an ellipse."
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result."
```

See Also Ellipse method

EllipseGraphic object

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

EllipticalArcTo Method

Syntax *GraphicBuilder*.**EllipticalArcTo**(*X* As Double, *Y* As Double, *ControlX* As Double, *ControlY* As Double, *Angle* As Double, [*Eccentricity* As Double])

Description The EllipticalArcTo method draws an elliptical arc as part of a path within a BeginPath/EndPath block. The resulting Graphic "type" can vary depending on what you draw, and how you draw it.

The EllipticalArcTo method can be used at any point to draw the next next segment of a path. The arc is drawn from the current point to the point defined by the method's X and Y arguments. The current position is then updated to be the end of the arc.The arc is drawn using the current pen, and is not filled.

The Control point of the elliptical arc always lies on the arc. It partly controls the size of the full ellipse. A control point that is a large distance away, relative to the beginning and ending coordinates of the arc, defines a larger elliptical arc.

The *X* argument specifies the X coordinate position of the end point of the arc.

The *Y* argument specifies the Y coordinate position of the end point of the arc.

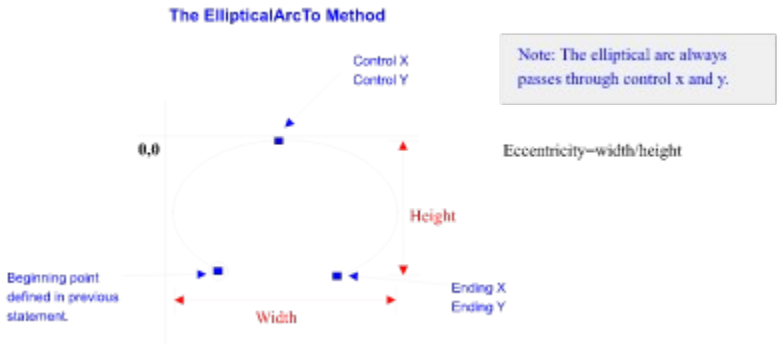
The *ControlX* argument specifies the X coordinate position of the arc's control point.

The *ControlY* argument specifies the Y coordinate position of the arc's control point.

The *Angle* argument specifies the angle to rotate the ellipse that draws the arc in a clockwise direction. The value is in degrees, and valid values are 0-359. This argument is optional.

The *Eccentricity* argument is optional and specifies the width/height ratio of the ellipse used to draw the arc.

The following illustration shows how the EllipticalArcTo method is used, and what its arguments represent.



Example The following example draws a group of elliptical arcs and prompts the user for the angle value.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
Dim dEllipseAngle As Double
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create a shape in the active diagram.
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object.
Set igxGraphic = igxShape.Graphic
' Begin a path to create the graphic
igxGrfxBuilder.BeginPath
```

```
' Prompt for an ellipse angle value
dEllipseAngle = Val(InputBox("Ellipse angle:"))
' Move the pen to 1.5, 1.5 in the coordinate space
igxGrfxBuilder.MoveTo 1.5, 1.5
igxGrfxBuilder.EllipticalArcTo 2.5, 1.5, 2, 0.5, dEllipseAngle, 0.5
igxGrfxBuilder.EllipticalArcTo 2.5, 2.5, 3.5, 2, dEllipseAngle, 2
igxGrfxBuilder.EllipticalArcTo 1.5, 2.5, 2, 3.5, dEllipseAngle, 0.5
igxGrfxBuilder.EllipticalArcTo 1.5, 1.5, 0.5, 2, dEllipseAngle, 2
igxGrfxBuilder.EllipticalArcTo 2.5, 1.5, 2, 0, dEllipseAngle, 0.5
igxGrfxBuilder.EllipticalArcTo 2.5, 2.5, 4, 2, dEllipseAngle, 2
igxGrfxBuilder.EllipticalArcTo 1.5, 2.5, 2, 4, dEllipseAngle, 0.5
igxGrfxBuilder.EllipticalArcTo 1.5, 1.5, 0, 2, dEllipseAngle, 2
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "Done. Click OK to continue."
```

EndClipPath Method

Syntax *GraphicBuilder*.**EndClipPath**

Description The EndClipPath method ends the definition of a clipping path. You begin defining a clipping path by calling the BeginClipPath method. A clipping path is used to define a closed graphic that is used in a Boolean Intersection operation (see the BeginClipPath method for details).

Example Refer to the Example for the BeginClipPath method.

See Also BeginClipPath method

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

EndPath Method

Syntax *GraphicBuilder*.**EndPath**

Description The EndPath method is used in conjunction with the BeginPath method to draw polygons with the GraphicBuilder object, using a point-by-point method. Anything you can draw with a mouse may be done with this approach. You can use any valid GraphicBuilder method to draw inside a BeginPath/EndPath bracket. The resulting Graphic “type” can vary depending on what you draw, and how you draw it.

For more information about using the BeginPath and EndPath methods for drawing graphics, refer to the BeginPath method.

Example Refer to the Example given for the BeginPath method.

See Also BeginPath method

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Graphic Property

Syntax *GraphicBuilder*.**Graphic**

Data Type Graphic object (read-only, See Object Properties)

Description The Graphic property returns a Graphic object that represents the graphic in the GraphicBuilder object. This is the way the graphic created with the GraphicBuilder can be transferred to shape or TextGraphicObject for use in a diagram.

The Graphic object constructed with the GraphicBuilder can be any of the valid graphic primitive, or a GraphicGroup object. If you draw more than one graphic, each graphic is placed in the Graphics collection (Graphic.GraphicGroup.Graphics). The Reset method clears any objects are being held in the Graphics collection

Example The following example creates a shape on the active diagram and then replaces the shape's graphic with a pentagon graphic created using the GraphicsBuilder object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a pentagon to the graphic
igxGrfxBuilder.Polygon 0.5, 0.5, 0.5, 5, 45
' Display a message box before replacing shape
MsgBox "Click OK to replace shape with an pentagon."
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
```

See Also Graphic object

DiagramObjects.AddGraphic method

iGrafx API Object Hierarchy

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Height Property

Syntax *GraphicBuilder*.**Height**

Data Type Double (read/write)

Description The Height property specifies the height of the coordinate space used by the GraphicBuilder to construct a graphic. For example, if the Width and Height properties are set to 2.0, the all arguments for GraphicBuilder methods that work within the relative coordinate space can use values between 0.0 and 2.0 to specify points that are inside of the coordinate space boundaries.

Note that specifying values for the arguments of GraphicBuilder methods that exceed the coordinate space boundaries is allowed, as well as using negative values.

Example The following example adds a shape in the active diagram. It then uses the GraphicBuilder to draw a diamond-shaped polygon whose points range from 0.0 to 3.0 in the coordinate space, and replaces the shape's graphic with this polygon. Then a second shape is added, the Height and Width properties of the GraphicBuilder are changed, and then a new polygon is drawn using the same coordinates. This new polygon is used to replace the graphic of the second shape. Then both shapes are selected so you can see how the graphic size relates to the size of the bounding box of the shapes.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create first polygon
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 1.5, 0
igxGrfxBuilder.LineTo 3, 1
igxGrfxBuilder.LineTo 1.5, 2
igxGrfxBuilder.LineTo 0, 1
' Close the polygon so it can be filled
igxGrfxBuilder.Close
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to green
igxShape.FillColor = vbGreen
MsgBox "View the diagram"
' Create a second shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Increase the size of the GraphicBuilder coordinate space
```

```
igxGrfxBuilder.Reset
igxGrfxBuilder.Height = 2
igxGrfxBuilder.Width = 3
' Draw the same polygon as the first time
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 1.5, 0
igxGrfxBuilder.LineTo 3, 1
igxGrfxBuilder.LineTo 1.5, 2
igxGrfxBuilder.LineTo 0, 1
' Close the polygon so it can be filled
igxGrfxBuilder.Close
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to green
igxShape.FillColor = vbRed
MsgBox "View the diagram"
' Select the two shapes so their bounding rectangles are visible
' and display message boxes for the user
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
    ActiveDiagram.DiagramObjects.Item(iCount).Selected = True
Next iCount
MsgBox "Compare the size of the graphic to the size" _
    & Chr(13) & "of the bounding box as a result of the" _
    & Chr(13) & "Height and Width property settings."
```

See Also Width property

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

LineTo Method

Syntax *GraphicBuilder*.**LineTo**(*X* As Double, *Y* As Double)

Description The LineTo method creates a line from the current point to the point defined by the *X* and *Y* arguments. Use this method to draw line segments. Using the MoveTo method sets the current point.

The *X* argument is typically a value from 0.0 to 1.0 that specifies the horizontal position of the point to which the line is drawn within the GraphicBuilder's coordinate space.

The *Y* argument is typically a value from 0.0 to 1.0 that specifies the vertical position of the point to which the line is drawn within the GraphicBuilder's coordinate space.

Note that the coordinate space used by the GraphicBuilder object is defined by the Height and Width properties.

Example The following example creates a shape on the active diagram, and then replaces the graphic of the shape with the graphic in the GraphicsBuilder object. The GraphicsBuilder object uses the MoveTo and LineTo methods to create a polygon graphic.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create first polygon
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 0.4, 1
igxGrfxBuilder.LineTo 0.4, 0.3
igxGrfxBuilder.LineTo 0.6, 0.3
igxGrfxBuilder.LineTo 0.6, 1
igxGrfxBuilder.LineTo 0.4, 1
' Close the polygon so it can be filled
igxGrfxBuilder.Close
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Create path for other polygon
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 0.4, 0.2
igxGrfxBuilder.LineTo 0.4, 0
igxGrfxBuilder.LineTo 0.6, 0
igxGrfxBuilder.LineTo 0.6, 0.2
igxGrfxBuilder.LineTo 0.4, 0.2
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
```

```
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to green
igxShape.FillColor = vbGreen
```

See Also

Height property

MoveTo method

Width property

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

MoveTo Method

Syntax *GraphicBuilder*.**MoveTo**(*X* As Double, *Y* As Double)

Description The MoveTo method moves the drawing position from the current point to the point defined by the supplied arguments *X* and *Y*. No line is drawn when this method is used; just the drawing point is moved.

The *X* argument is typically a value from 0.0 to 1.0 that specifies the horizontal position of the point that is being moved to within the GraphicBuilder's coordinate space.

The *Y* argument is typically a value from 0.0 to 1.0 that specifies the vertical position of the point that is being moved to within the GraphicBuilder's coordinate space.

Note that the coordinate space used by the GraphicBuilder object is defined by the Height and Width properties.

Example The following example creates a shape on the active diagram, and then replaces the graphic of the shape with the graphic in the GraphicsBuilder object. The GraphicsBuilder object uses the MoveTo and LineTo methods to create the polygon graphic.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create first polygon
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 0.4, 1
igxGrfxBuilder.LineTo 0.4, 0.3
igxGrfxBuilder.LineTo 0.6, 0.3
igxGrfxBuilder.LineTo 0.6, 1
igxGrfxBuilder.LineTo 0.4, 1
' Close the polygon so it can be filled
igxGrfxBuilder.Close
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Create path for other polygon
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 0.4, 0.2
igxGrfxBuilder.LineTo 0.4, 0
igxGrfxBuilder.LineTo 0.6, 0
igxGrfxBuilder.LineTo 0.6, 0.2
igxGrfxBuilder.LineTo 0.4, 0.2
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
```

```
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to green
igxShape.FillColor = vbGreen
```

See Also

Height property

LineTo method

Width property

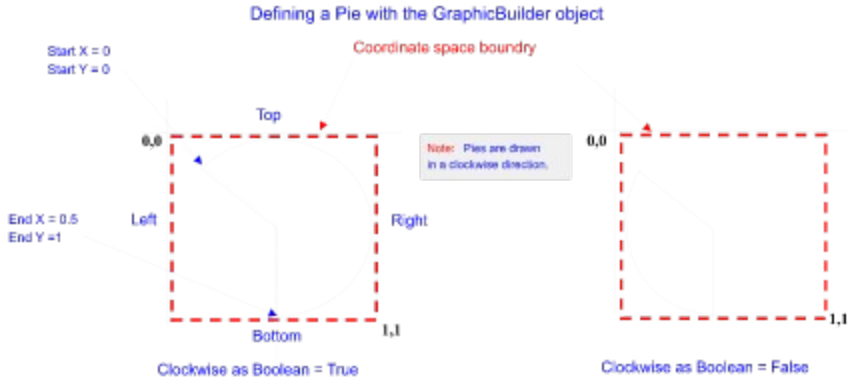
{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Pie Method

Syntax *GraphicBuilder*.**Pie**(*Left* As Double, *Top* As Double, *Right* As Double, *Bottom* As Double, *StartX* As Double, *StartY* As Double, *EndX* As Double, *EndY* As Double, *Clockwise* As Boolean)

Description The Pie method draws a pie-shaped wedge bounded by the intersection of an ellipse and two radials. The resulting graphic is an ArcGraphic object, with the ArcType property set to ixArcPie. The pie is outlined using the current pen, and is filled using the current brush. The current position is neither used nor updated by the Pie method.

The following illustration shows how to use the Pie method.



The arguments for this method are as follows:

The *Left* argument specifies the position along the X coordinate axis of the upper left corner of the bounding rectangle of the pie.

The *Top* argument specifies the position along the Y coordinate axis of the upper left corner of the bounding rectangle of the pie.

The *Right* argument specifies the position along the X coordinate axis of the lower right corner of the bounding rectangle of the pie.

The *Bottom* argument specifies the position along the Y coordinate axis of the lower right corner of the bounding rectangle of the pie.

The *XRadial1* argument specifies the position along the X coordinate axis of the end point of the first radial.

The *YRadial1* argument specifies the position along the Y coordinate axis of the end point of the first radial.

The *XRadial2* argument specifies the position along the X coordinate axis of the end point of the second radial.

The *YRadial2* argument specifies the position along the X coordinate axis of the end point of the second radial.

The points defined by the Left, Top, Right, and Bottom arguments specify the bounding rectangle of the pie. An ellipse formed by the bounding rectangle defines the curve of the pie. The curve begins at the point where the ellipse intersects the first radial, and extends counterclockwise to the point where the ellipse intersects the second radial. (A radial is a line segment drawn from the center of an ellipse to a specified endpoint on an ellipse.)

Example The following example places two shapes in the active diagram. The Pie method is then used to draw a graphic with the GraphicBuilder. The Pie graphic is used to replace the graphic in the first shape. The GraphicBuilder is then reset, and another Pie is drawn, which is then used to replace Shape 2's graphic.

```
' Dimension the variables
Dim igxShape1 As Shape
```

```
Dim igxShape2 As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create two shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "Shape added to diagram. Replace the shape graphics" _
    & Chr(13) & "with new graphics from the GraphicBuilder."
' Create a Pie graphic with the GraphicBuilder
igxGrfxBuilder.Pie 0.25, 0.25, 1, 1, 0, 0.5, 1, 0.7, True
' Replace the graphic inside the first shape with the new graphic
igxShape1.Graphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
' Reset the GraphicBuilder and draw a new shape
igxGrfxBuilder.Reset
' Create another Pie graphic with the GraphicBuilder
igxGrfxBuilder.Pie 0.3, 0, 0.8, 0.9, 1, 0.6, 0.4, 0, False
' Replace the graphic inside the second shape with the new graphic
igxShape2.Graphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
```

See Also

Arc method

CircularArcTo method

EllipticalArcTo method

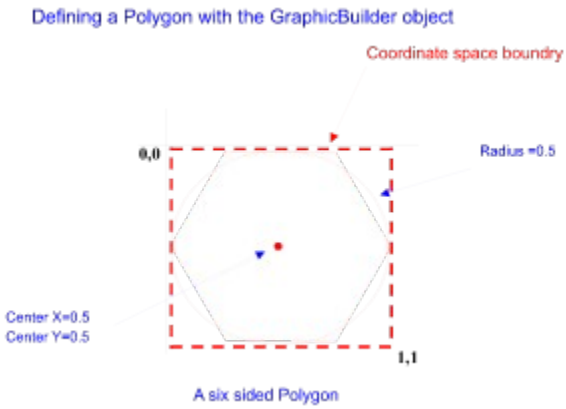
{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Polygon Method

Syntax *GraphicBuilder*.**Polygon**(*CenterX* As Double, *CenterY* As Double, *Radius* As Double, *Points* As Integer, [*StartAngle* As Double])

Description The Polygon method is used to create a closed polygon graphic, such as a pentagon or triangle.

The following illustration shows the elements of creating a polygon with the GraphicBuilder.



The *CenterX* argument is a value from 0.0 to 1.0 that specifies the horizontal center position of the polygon graphic.

The *CenterY* argument is a value from 0.0 to 1.0 that specifies the vertical center position of the polygon graphic.

The *Radius* argument is a value from 0.0 to 1.0 that specifies the outer radius of the polygon graphic.

The *Points* argument is an integer value that specifies how many points to create for the polygon. For example, set this value to 5 to create a pentagon.

The *StartAngle* argument is a value from 0 to 359 that specifies the rotation to be applied to the polygon graphic when it is created.

Example The following example creates a shape on the active diagram, and then replaces the shape's graphic with a pentagon graphic created using the GraphicsBuilder object.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a pentagon to the graphic
igxGrfxBuilder.Polygon 0.5, 0.5, 0.5, 5, 45
' Display a message box before replacing shape
```

```
MsgBox "Click OK to replace shape with an pentagon."
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
```

See Also PolygonGraphic object

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Rectangle Method

Syntax *GraphicBuilder*.**Rectangle**(*Left* As Double, *Top* As Double, *Width* As Double, *Height* As Double, [*Rounding* As Double])

Description The Rectangle method is used to create a rectangle graphic.

The *Left* argument is a value from 0.0 to 1.0 that specifies the position of the left side of the rectangle graphic.

The *Top* argument is a value from 0.0 to 1.0 that specifies the position of the top of the rectangle graphic.

The *Width* argument is a value from 0.0 to 1.0 that specifies the width of the rectangle graphic.

The *Height* argument is a value from 0.0 to 1.0 that specifies the height of the rectangle graphic.

Example The following example creates a shape on the active diagram, and then replaces the shape's graphic with a RectangleGraphic created using the GraphicsBuilder object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a rectangle to the graphic
igxGrfxBuilder.Rectangle 0.1, 0.1, 0.8, 0.8
' Display a message box before replacing shape
MsgBox "Click OK to replace shape with a rectangle."
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
```

See Also RectangleGraphic object

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Reset Method

Syntax *GraphicBuilder*.**Reset**

Description The Reset method clears all previous drawing from the GraphicBuilder, allowing you to start drawing a completely new graphic. For instance, if you draw two polygons, normally they are part of the same graphic. If you want two separate polygons, use the Reset method in between the definitions of the two polygons.

There are three ways you can start fresh drawing a graphic with the GraphicBuilder.

- Use the Reset method
- Set your GraphicBuilder variable (i.e. igxGrfxBuilder) to a "new" GraphicBuilder object, as follows: `Set igxGrfxBuilder = New GraphicBuilder`
- Dimension a "new" GraphicBuilder object

The first choice is the preferred, and recommended, way of starting a new graphic.

Example The following example illustrates how to use the Reset method when using the GraphicBuilder object. Two shapes are placed in the diagram. Then the GraphicBuilder is used to draw a polygon. The polygon is used to replaced the graphic in the first shape. Then the GraphicBuilder is reset to start a new graphic—a teardrop drawn with the BSplineTo method. This graphic is then used to replace the graphic in the second shape.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create two shapes in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3, Application.ShapeLibraries.Item(1).Item(4))
MsgBox "Shape added to diagram. Replace the shape graphics" _
    & Chr(13) & "with new graphics from the GraphicBuilder."
' Create a graphic with the GraphicBuilder
igxGrfxBuilder.Polygon 0.5, 0.5, 0.5, 6, 45
' Replace the graphic inside the first shape with the new graphic
igxShape1.Graphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
' Reset the GraphicBuilder and draw a new shape
igxGrfxBuilder.Reset
igxGrfxBuilder.BeginPath
igxGrfxBuilder.MoveTo 0.5, 0
igxGrfxBuilder.BSplineTo 0.75, 0.5
igxGrfxBuilder.BSplineTo 0.5, 1
igxGrfxBuilder.BSplineTo 0.25, 0.5
igxGrfxBuilder.BSplineTo 0.5, 0
igxGrfxBuilder.Close
igxGrfxBuilder.EndPath
' Replace the graphic inside the second shape with the new graphic
igxShape2.Graphic.Replace igxGrfxBuilder.Graphic
MsgBox "View the result"
```

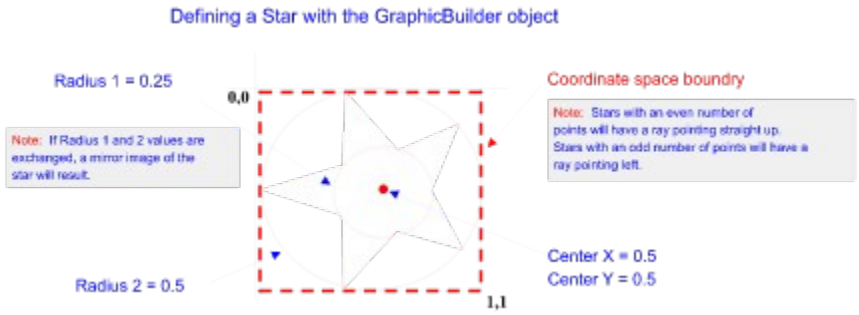
{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Star Method

Syntax *GraphicBuilder*.**Star**(*CenterX* As Double, *CenterY* As Double, *Radius1* As Double, *Radius2* As Double, *NumberOfStarPoints* As Integer, [*StartAngle* As Double])

Description The Star method is used to create a polygon star graphic. The arguments to the method provide the following data.

The following illustration shows how to create a star using the GraphicBuilder.



The *CenterX* argument is a value from 0.0 to 1.0 that specifies the horizontal center position of the star graphic.

The *CenterY* argument is a value from 0.0 to 1.0 that specifies the vertical center position of the star graphic.

The *Radius1* argument is a value from 0.0 to 1.0 that specifies the outer radius of the star graphic. The outer radius defines the location of the tips of the star points.

The *Radius2* argument is a value from 0.0 to 1.0 that specifies the inner radius of the star graphic. The inner radius is where the points touch the inner part of the star.

The *NumberOfStarPoints* argument is an integer value that specifies the number of points on the star.

The *StartAngle* argument is a value from 0 to 359 that specifies the rotation to be applied to the star graphic when it is created.

Example The following example creates a shape on the active diagram, and then replaces the shape's graphic with a polygon star graphic created using the GraphicsBuilder object.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape on the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Add a star to the graphic
igxGrfxBuilder.Star 0.5, 0.5, 0.5, 0.1, 5, 30
' Display a message box before replacing shape
MsgBox "Click OK to replace shape with a star."
```

```
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
```

See Also PolygonGraphic object

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

Width Property

Syntax *GraphicBuilder*.**Width**

Data Type Double (read/write)

Description The Width property specifies the width of the coordinate space used by the GraphicBuilder to construct a graphic. For example, if the Width and Height properties are set to 2.0, the all arguments for GraphicBuilder methods that work within the relative coordinate space can use values between 0.0 and 2.0 to specify points that are inside of the coordinate space boundaries.

Note that specifying values for the arguments of GraphicBuilder methods that exceed the coordinate space boundaries is allowed, as well as using negative values.

Example The following example adds a shape in the active diagram. It then uses the GraphicBuilder to draw a diamond-shaped polygon whose points range from 0.0 to 3.0 in the coordinate space, and replaces the shape's graphic with this polygon. Then a second shape is added, the Height and Width properties of the GraphicBuilder are changed, and then a new polygon is drawn using the same coordinates. This new polygon is used to replace the graphic of the second shape. Then both shapes are selected so you can see how the graphic size relates to the size of the bounding box of the shapes.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxGraphic As Graphic
' Dimension the new GraphicBuilder object
Dim igxGrfxBuilder As New GraphicBuilder
' Create the shape on the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Begin a path to create first polygon
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 1.5, 0
igxGrfxBuilder.LineTo 3, 1
igxGrfxBuilder.LineTo 1.5, 2
igxGrfxBuilder.LineTo 0, 1
' Close the polygon so it can be filled
igxGrfxBuilder.Close
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to green
igxShape.FillColor = vbGreen
MsgBox "View the diagram"
' Create a second shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440 * 3, Application.ShapeLibraries.Item(1).Item(1))
MsgBox "View the diagram"
' Get the Graphic object from the shape object
Set igxGraphic = igxShape.Graphic
' Increase the size of the GraphicBuilder coordinate space
```

```
igxGrfxBuilder.Reset
igxGrfxBuilder.Height = 2
igxGrfxBuilder.Width = 3
' Draw the same polygon as the first time
igxGrfxBuilder.BeginPath
' Build the polygon
igxGrfxBuilder.MoveTo 1.5, 0
igxGrfxBuilder.LineTo 3, 1
igxGrfxBuilder.LineTo 1.5, 2
igxGrfxBuilder.LineTo 0, 1
' Close the polygon so it can be filled
igxGrfxBuilder.Close
' End the path to make this a separate polygon
igxGrfxBuilder.EndPath
' Replace the graphic inside the shape with the new graphic
igxGraphic.Replace igxGrfxBuilder.Graphic
' Set the fill color of the shape to green
igxShape.FillColor = vbRed
MsgBox "View the diagram"
' Select the two shapes so their bounding rectangles are visible
' and display message boxes for the user
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
    ActiveDiagram.DiagramObjects.Item(iCount).Selected = True
Next iCount
MsgBox "Compare the size of the graphic to the size" _
    & Chr(13) & "of the bounding box as a result of the" _
    & Chr(13) & "Height and Width property settings."
```

See Also Height property

{button GraphicBuilder object,JI(`igrafxrf.HLP',`GraphicBuilder_Object')}

GeometryHelper Object

The GeometryHelper object provides methods and properties that can help in the creation of graphics or in other aspects of drawing diagrams where accurate geometric data is needed. Most standard geometric functions are available, such as Sin, Cos, and ArcTan, the value of pi, and conversion between degrees and radians and vice versa. The GeometryHelper object is accessed from the Application object.

Properties, Methods, and Events

All of the properties, methods, and events for the GeometryHelper object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Angle	
Parent	ArcTan	
pi	ArcTan2	
	Cos	
	DegreesToRadians	
	Distance	
	RadiansToDegrees	
	RCos	
	RSin	
	Sin	

Example

The following example creates a shape, and then replaces it's graphic with a new geometric pattern. It uses several of the GeometryHelper methods to perform calculations used to draw lines using a GraphicHelper object. After the shape is created, the adjustment point can be moved to change the pattern.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    igxShape1.DiagramObject.Width = 3000
    igxShape1.DiagramObject.Height = 3000
    igxShape1.Adjustments.Add 0.1, 0.9
    igxShape1.Adjustments.Add 0, 1
    igxShape1.Adjustments.Add 1, 0
    igxShape1.Adjustments.Add 0.9, 0.9
    Generate igxShape1
    MsgBox "Try moving the adjustment point."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
    ' Redraw the geometric shape if an adjustment point is moved
    Generate igxShape1
```

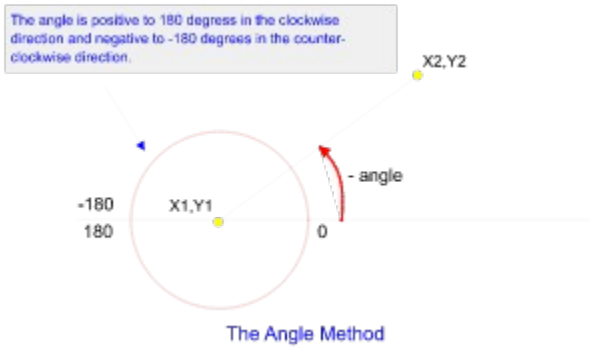
```
End Sub

' Generate the geometric pattern
Public Sub Generate(s As Shape)
' Bail out if anything goes wrong
On Error GoTo ErrorHandler
    ' Dimension the variables
    Dim i As Integer
    Dim repetitions As Integer
    Dim adjs As Integer
    Dim geom As GeometryHelper
    Dim builder As GraphicBuilder
    Dim movedone As Boolean
    ' Make a new GraphicBuilder
    Set builder = New GraphicBuilder
    movedone = False
    ' Get a GeometryHelper object
    Set geom = Application.GeometryHelper
    repetitions = 8
    ' Get the number of adjustment points
    adjs = s.Adjustments.Count
    ' Begin a geometric path
    builder.BeginPath
    For i = 1 To repetitions
        Dim rotateby As Double
        Dim adj As Adjustment
        Dim reps As Double
        reps = repetitions
        rotateby = (i - 1) * 360# / reps
        For Each adj In s.Adjustments
            Dim xRep As Double
            Dim yRep As Double
            Dim radius As Double
            Dim angle As Double
            xRep = adj.X
            yRep = adj.Y
            ' Use the GeometryHelper to calculate ditance and angle
            radius = geom.Distance(0.5, 0.5, xRep, yRep)
            angle = geom.angle(0.5, 0.5, xRep, yRep)
            angle = angle + rotateby
            ' Use the GeometryHelper to calulate the point coordinates
            ' based on radius and angle, using RCos and RSin
            xRep = 0.5 + geom.RCos(radius, angle)
            yRep = 0.5 + geom.RSin(radius, angle)
            If movedone Then
                ' Draw a line
                builder.LineTo xRep, yRep
            Else
                ' Move the pen without drawing
                builder.MoveTo xRep, yRep
```

```
                movedone = True
            End If
        Next adj
    Next i
    builder.Close
    ' Close the geometric path
    builder.EndPath
    ' Replace the shape graphic with the new graphic
    s.Graphic.Replace builder.Graphic
' Bail out if anything goes wrong
ErrorHandler:
    Exit Sub
End Sub
```

Angle Method

Syntax *GeometryHelper.***Angle**(*X1* As Double, *Y1* As Double, *X2* As Double, *Y2* As Double) As Double

Description The Angle method returns the angle between the two points supplied in the four arguments X1, Y1, X2, and Y2. The two points create an imaginary line. The Angle method returns the angle of the line between the two points and a horizontal reference line that intersects the first point, as shown in the following illustration.



The Angle Method

Example The following example creates a shape with two adjustment points. By moving the adjustment points, the angle of the line changes, showing the result of the angle method.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
   igxShape1.DiagramObject.Width = 3000
   igxShape1.DiagramObject.Height = 3000
   igxShape1.Adjustments.Add 0.1, 0.1
   igxShape1.Adjustments.Add 0.9, 0.9
   Generate igxShape1
   ActiveDiagram.Selection.AddAll ixObjectShape
   MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
   ' Redraw the geometric shape if an adjustment point is moved
   Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(igxShape As Shape)
   ' Dimension the variables
   Dim igxBuilder As New GraphicBuilder
   Dim igxHelper As GeometryHelper
   Dim X1 As Double, Y1 As Double
   Dim X2 As Double, Y2 As Double
   Set igxHelper = GeometryHelper
```

```
    X1 = igxShape.Adjustments.Item(1).X
    Y1 = igxShape.Adjustments.Item(1).Y
    X2 = igxShape.Adjustments.Item(2).X
    Y2 = igxShape.Adjustments.Item(2).Y
    ' Draw a straight line from one point to the other
    With igxBuilder
       .BeginPath
       .MoveTo X1, Y1
       .LineTo X2, Y2
       .EndPath
    End With
    igxShape.Graphic.Replace igxBuilder.Graphic
    igxShape.Text = "Angle = " & igxHelper.Angle(X1, Y1, X2, Y2)
End Sub
```

See Also Distance method

{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

ArcTan Method

Syntax *GeometryHelper*.**ArcTan**(*YOverX* As Double) As Double

Description The ArcTan method returns the ArcTan for the supplied *YOverX* argument. ArcTan is an angle based on the ratio of the Y coordinate divided by the X coordinate.

Example The following example creates a shape with two adjustment points. By moving the adjustment points, the angle of the line changes, showing the result of the ArcTan method.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
   igxShape1.DiagramObject.Width = 3000
   igxShape1.DiagramObject.Height = 3000
   igxShape1.Adjustments.Add 0.1, 0.1
   igxShape1.Adjustments.Add 0.9, 0.9
   Generate igxShape1
   ActiveDiagram.Selection.AddAll ixObjectShape
   MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
   ' Redraw the geometric shape if an adjustment point is moved
   Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(igxShape As Shape)
   ' Dimension the variables
   Dim igxBuilder As New GraphicBuilder
   Dim igxHelper As GeometryHelper
   Dim X1 As Double, Y1 As Double
   Dim X2 As Double, Y2 As Double
   Set igxHelper = GeometryHelper
   X1 = igxShape.Adjustments.Item(1).X
   Y1 = igxShape.Adjustments.Item(1).Y
   X2 = igxShape.Adjustments.Item(2).X
   Y2 = igxShape.Adjustments.Item(2).Y
   ' Draw a straight line from one point to the other
   With igxBuilder
      .BeginPath
      .MoveTo X1, Y1
      .LineTo X2, Y2
      .EndPath
   End With
   igxShape.Graphic.Replace igxBuilder.Graphic
   igxShape.Text = "ArcTan Angle = " & _
       igxHelper.ArcTan((Y2 - Y1) / (X2 - X1))
```

```
End Sub
```

See Also ArcTan2 method

{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

ArcTan2 Method

Syntax *GeometryHelper*.**ArcTan2**(*Y* As Double, *X* As Double) As Double

Description The ArcTan2 method returns the ArcTan for the supplied arguments Y and X. The ArcTan2 method returns an angle based on the position of Y and X from the origin.

Example The following example creates a shape with two adjustment points. By moving the adjustment points, the angle of the line changes, showing the result of the ArcTan2 method.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
   igxShape1.DiagramObject.Width = 3000
   igxShape1.DiagramObject.Height = 3000
   igxShape1.Adjustments.Add 0.1, 0.1
   igxShape1.Adjustments.Add 0.9, 0.9
   Generate igxShape1
   ActiveDiagram.Selection.AddAll ixObjectShape
   MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
   ' Redraw the geometric shape if an adjustment point is moved
   Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(igxShape As Shape)
   ' Dimension the variables
   Dim igxBuilder As New GraphicBuilder
   Dim igxHelper As GeometryHelper
   Dim X1 As Double, Y1 As Double
   Dim X2 As Double, Y2 As Double
   Set igxHelper = GeometryHelper
   X1 = igxShape.Adjustments.Item(1).X
   Y1 = igxShape.Adjustments.Item(1).Y
   X2 = igxShape.Adjustments.Item(2).X
   Y2 = igxShape.Adjustments.Item(2).Y
   ' Draw a straight line from one point to the other
   With igxBuilder
      .BeginPath
      .MoveTo X1, Y1
      .LineTo X2, Y2
      .EndPath
   End With
   igxShape.Graphic.Replace igxBuilder.Graphic
   igxShape.Text = "ArcTan2 Angle = " & _
       igxHelper.ArcTan2(Y2 - Y1, X2 - X1)
```

```
End Sub
```

See Also ArcTan method

{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

Cos Method

Syntax *GeometryHelper*.**Cos**(*AngleInDegrees* As Double) As Double

Description The Cos method returns the Cosine of the angle (the X coordinate) supplied in the *AngleInDegrees* argument. The Cosine is a ratio derived from the length of the adjacent leg, divided by the length of the hypotenuse of a right triangle.

Example The following example creates a shape with two adjustment points. By moving the adjustment points, the angle of the line changes, showing the result of the Cos and Sin methods.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
    igxShape1.DiagramObject.Width = 3000
    igxShape1.DiagramObject.Height = 3000
    igxShape1.Adjustments.Add 0.1, 0.1
    igxShape1.Adjustments.Add 0.9, 0.9
    Generate igxShape1
    ActiveDiagram.Selection.AddAll ixObjectShape
    MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
    ' Redraw the geometric shape if an adjustment point is moved
    Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(shp As Shape)
    Dim igxBuilder As New GraphicBuilder
    Dim igxHelper As GeometryHelper
    Dim X1 As Double, Y1 As Double
    Dim X2 As Double, Y2 As Double
    Set igxHelper = GeometryHelper
    X1 = shp.Adjustments.Item(1).X
    Y1 = shp.Adjustments.Item(1).Y
    X2 = shp.Adjustments.Item(2).X
    Y2 = shp.Adjustments.Item(2).Y
    ' Draw a straight line from one point to the other
    With igxBuilder
       .BeginPath
       .MoveTo X1, Y1
       .LineTo X2, Y2
       .EndPath
    End With
    shp.Graphic.Replace igxBuilder.Graphic
    ' Display the result of the Cos and Sin methods
    shp.Text = "Cosine = " & _
```

```
        Round(igxHelper.Cos(igxHelper.Angle(X1, Y1, X2, Y2)), 2) _
        & Chr(13) & "Sine = " & _
        Round(igxHelper.Sin(igxHelper.Angle(X1, Y1, X2, Y2)), 2)
End Sub
```

See Also

RCos method

RSin method

Sin method

{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

DegreesToRadians Method

Syntax *GeometryHelper*.**DegreesToRadians**(*Degrees* As Double) As Double

Description The DegreesToRadians method takes a value in degrees and returns the value converted to radians (360 degrees = 2 * pi radians).

Note Sin, Cos, RSin, and RCos methods accept values in degrees only.

Example The following example creates a shape with two adjustment points. By moving the adjustment points, the angle of the line changes, showing the result of the Angle method, converted to Radians.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
    igxShape1.DiagramObject.Width = 3000
    igxShape1.DiagramObject.Height = 3000
    igxShape1.Adjustments.Add 0.1, 0.1
    igxShape1.Adjustments.Add 0.9, 0.9
    Generate igxShape1
    ActiveDiagram.Selection.AddAll ixObjectShape
    MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
    ' Redraw the geometric shape if an adjustment point is moved
    Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(shp As Shape)
    Dim igxBuilder As New GraphicBuilder
    Dim igxHelper As GeometryHelper
    Dim X1 As Double, Y1 As Double
    Dim X2 As Double, Y2 As Double
    Set igxHelper = GeometryHelper
    X1 = shp.Adjustments.Item(1).X
    Y1 = shp.Adjustments.Item(1).Y
    X2 = shp.Adjustments.Item(2).X
    Y2 = shp.Adjustments.Item(2).Y
    ' Draw a straight line from one point to the other
    With igxBuilder
       .BeginPath
       .MoveTo X1, Y1
       .LineTo X2, Y2
       .EndPath
    End With
    shp.Graphic.Replace igxBuilder.Graphic
```

```
    ' Round off the value and display it
    shp.Text = "Angle in Radians = " & _
       Round(igxHelper.DegreesToRadians _
       (igxHelper.Angle(X1, Y1, X2, Y2)), 2)
End Sub
```

See Also RadiansToDegrees method

{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

Distance Method

Syntax *GeometryHelper*.**Distance** (*X1* As Double, *Y1* As Double, *X2* As Double, *Y2* As Double) As Double

Description The Distance method returns the distance between the two points supplied in the four arguments, X1, Y1, X2, and Y2.

Example The following example creates a shape with two adjustment points. By moving the adjustment points, the length of the line changes, showing the result of the Distance method.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
    igxShape1.DiagramObject.Width = 3000
    igxShape1.DiagramObject.Height = 3000
    igxShape1.Adjustments.Add 0.1, 0.1
    igxShape1.Adjustments.Add 0.9, 0.9
    Generate igxShape1
    ActiveDiagram.Selection.AddAll ixObjectShape
    MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
    ' Redraw the geometric shape if an adjustment point is moved
    Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(shp As Shape)
    Dim igxBuilder As New GraphicBuilder
    Dim igxHelper As GeometryHelper
    Dim X1 As Double, Y1 As Double
    Dim X2 As Double, Y2 As Double
    Set igxHelper = GeometryHelper
    X1 = shp.Adjustments.Item(1).X
    Y1 = shp.Adjustments.Item(1).Y
    X2 = shp.Adjustments.Item(2).X
    Y2 = shp.Adjustments.Item(2).Y
    ' Draw a straight line from one point to the other
    With igxBuilder
       .BeginPath
       .MoveTo X1, Y1
       .LineTo X2, Y2
       .EndPath
    End With
    shp.Graphic.Replace igxBuilder.Graphic
    ' Display the value
    shp.Text = "Length (local) = " & _
```

```
        Round(igxHelper.Distance(X1, Y1, X2, Y2), 2)
End Sub
```

See Also Angle method

{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

pi Property

Syntax *GeometryHelper*.**pi**

Data Type Double (read-only)

Description The pi property returns the value of pi to 14 decimal places. Pi is a constant derived from the circumference of a circle divided by it's radius.

{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

RadiansToDegrees Method

Syntax *GeometryHelper*.**RadiansToDegrees**(*Radians* As Double) As Double

Description The RadiansToDegrees method converts a value given in radians and returns an angle value in degrees (2 * pi radians = 360 degrees). The *Radians* argument is the value to convert.

Note Sin, Cos, RSin, and RCos methods accept values in degrees only.

Example The following example creates a shape with two adjustment points. By moving the adjustment points, the angle of the line changes, showing the result of the Angle method, converted from Radians to Degrees.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
    igxShape1.DiagramObject.Width = 3000
    igxShape1.DiagramObject.Height = 3000
    igxShape1.Adjustments.Add 0.1, 0.1
    igxShape1.Adjustments.Add 0.9, 0.9
    Generate igxShape1
    ActiveDiagram.Selection.AddAll ixObjectShape
    MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
    ' Redraw the geometric shape if an adjustment point is moved
    Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(shp As Shape)
    Dim igxBuilder As New GraphicBuilder
    Dim igxHelper As GeometryHelper
    Dim X1 As Double, Y1 As Double
    Dim X2 As Double, Y2 As Double
    Dim Radians As Double
    Set igxHelper = GeometryHelper
    X1 = shp.Adjustments.Item(1).X
    Y1 = shp.Adjustments.Item(1).Y
    X2 = shp.Adjustments.Item(2).X
    Y2 = shp.Adjustments.Item(2).Y
    ' Draw a straight line from one point to the other
    With igxBuilder
       .BeginPath
       .MoveTo X1, Y1
       .LineTo X2, Y2
       .EndPath
    End With
```

```
        shp.Graphic.Replace igxBuilder.Graphic
        ' Round off the value and display it
        Radians = igxHelper.DegreesToRadians _
           (igxHelper.Angle(X1, Y1, X2, Y2))
        shp.Text = "Angle in Degrees = " & _
           Round(igxHelper.RadiansToDegrees(Radians), 2)
    End Sub
```

See Also DegreesToRadians method

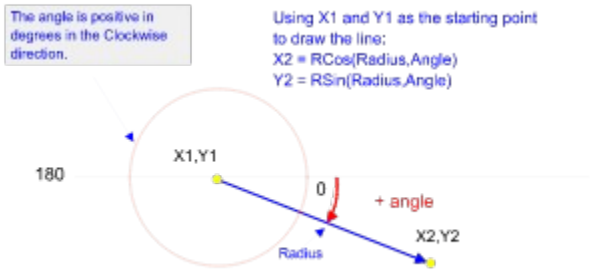
{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

RCos Method

Syntax *GeometryHelper*.**RCos** (*Radius* As Double, *AngleInDegrees* As Double) As Double

Description The RCos method returns the X coordinate of the endpoint of a line, based on the *Radius* and *AngleInDegrees* arguments, from an initial point X1, Y1. RCos is the X coordinate given a radius and an angle.

The following illustration shows how the method is used.



Definining Coordinates with Rsin and Rcos Methods

Example The following example creates a shape with two adjustment points. One point adjusts radius, and the other adjusts angle. When the adjustment points are moved with the mouse, the line is redrawn using the RCos and RSin methods.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

'Run this to create the new shape
Private Sub Main()
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
    igxShape1.DiagramObject.Width = 3000
    igxShape1.DiagramObject.Height = 3000
    igxShape1.Adjustments.Add 0.1, 0.7
    igxShape1.Adjustments.Add 0.1, 0.9
    Generate igxShape1
    ActiveDiagram.Selection.AddAll ixObjectShape
    MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
    If Index = 1 Then Y = 0.7
    If Index = 2 Then Y = 0.9
    ' Redraw the geometric shape if an adjustment point is moved
    Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(shp As Shape)
    Dim igxBuilder As New GraphicBuilder
    Dim igxHelper As GeometryHelper
    Dim X1 As Double, Y1 As Double
```

```
    Dim X2 As Double, Y2 As Double
    Dim Adj1X As Double, Adj2X As Double
    Set igxHelper = GeometryHelper
    X1 = 0.5
    Y1 = 0.5
    Adj1X = shp.Adjustments.Item(1).X
    Adj2X = shp.Adjustments.Item(2).X
    ' Calculate coordinate based on radius and angle
    X2 = igxHelper.RCos(Adj1X, Adj2X * 180) + 0.5
    Y2 = igxHelper.RSin(Adj1X, Adj2X * 180) + 0.5
    ' Draw a straight line from one point to the other
    With igxBuilder
       .BeginPath
       .MoveTo X1, Y1
       .LineTo X2, Y2
       .EndPath
    End With
    shp.Graphic.Replace igxBuilder.Graphic
    ' Round off the value and display it
    shp.Text = "Radius = " & Adj1X & Chr(13) & _
       "Angle = " & Adj2X * 360
End Sub
```

See Also

Cos method

RSin method

Sin method

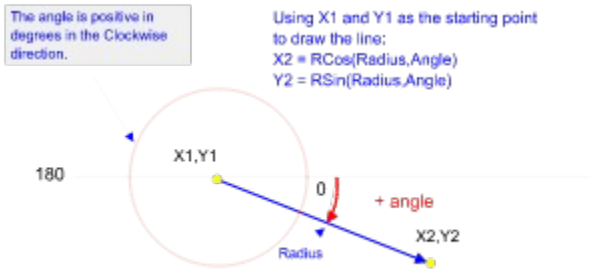
{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

RSin Method

Syntax *GeometryHelper*.**RSin**(*Radius* As Double, *AngleInDegrees* As Double) As Double

Description The RSin method returns the Y coordinate of the endpoint of a line, based on the *Radius* and *AngleInDegrees* arguments, from an initial point X1, Y1. RSin is the Y coordinate given a radius and an angle.

The following illustration shows how the method is used.



Definining Coordinates with Rsin and Rcos Methods

Example The following example creates a shape with two adjustment points. One point adjusts radius, and the other adjusts angle. When the adjustment points are moved with the mouse, the line is redrawn using the RCos and RSin methods.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
    igxShape1.DiagramObject.Width = 3000
    igxShape1.DiagramObject.Height = 3000
    igxShape1.Adjustments.Add 0.1, 0.7
    igxShape1.Adjustments.Add 0.1, 0.9
    Generate igxShape1
    ActiveDiagram.Selection.AddAll ixObjectShape
    MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
    If Index = 1 Then Y = 0.7
    If Index = 2 Then Y = 0.9
    ' Redraw the geometric shape if an adjustment point is moved
    Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(shp As Shape)
    Dim igxBuilder As New GraphicBuilder
    Dim igxHelper As GeometryHelper
    Dim X1 As Double, Y1 As Double
```

```
    Dim X2 As Double, Y2 As Double
    Dim Adj1X As Double, Adj2X As Double
    Set igxHelper = GeometryHelper
    X1 = 0.5
    Y1 = 0.5
    Adj1X = shp.Adjustments.Item(1).X
    Adj2X = shp.Adjustments.Item(2).X
    ' Calculate coordinate based on radius and angle
    X2 = igxHelper.RCos(Adj1X, Adj2X * 180) + 0.5
    Y2 = igxHelper.RSin(Adj1X, Adj2X * 180) + 0.5
    ' Draw a straight line from one point to the other
    With igxBuilder
       .BeginPath
       .MoveTo X1, Y1
       .LineTo X2, Y2
       .EndPath
    End With
    shp.Graphic.Replace igxBuilder.Graphic
    ' Round off the value and display it
    shp.Text = "Radius = " & Adj1X & Chr(13) & _
       "Angle = " & Adj2X * 360
End Sub
```

See Also

Cos method

RCos method

Sin method

{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

Sin Method

Syntax *GeometryHelper*.**Sin** (*AngleInDegrees* As Double) As Double

Description The Sin method returns the Sine of the angle supplied in the *AngleInDegrees* argument. Sine is a ratio derived from the length of the opposite leg divided by the legnth of the hypotenuse of a right triangle.

Example The following example creates a shape with two adjustment points. By moving the adjustment points, the angle of the line changes, showing the result of the Cos and Sin methods.

```
' Dimension a variable that hears shape events
Public WithEvents igxShape1 As Shape

' Run this to create the new shape
Private Sub Main()
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
    igxShape1.DiagramObject.Width = 3000
    igxShape1.DiagramObject.Height = 3000
    igxShape1.Adjustments.Add 0.1, 0.1
    igxShape1.Adjustments.Add 0.9, 0.9
    Generate igxShape1
    ActiveDiagram.Selection.AddAll ixObjectShape
    MsgBox "Try moving the adjustment points."
End Sub

' Do this every time an adjustment point is moved
Private Sub igxShape1_AdjustmentMove(ByVal Index As Integer, _
X As Double, Y As Double)
    ' Redraw the geometric shape if an adjustment point is moved
    Generate igxShape1
End Sub

' Generate the graphic for the shape
Public Sub Generate(shp As Shape)
    Dim igxBuilder As New GraphicBuilder
    Dim igxHelper As GeometryHelper
    Dim X1 As Double, Y1 As Double
    Dim X2 As Double, Y2 As Double
    Set igxHelper = GeometryHelper
    X1 = shp.Adjustments.Item(1).X
    Y1 = shp.Adjustments.Item(1).Y
    X2 = shp.Adjustments.Item(2).X
    Y2 = shp.Adjustments.Item(2).Y
    ' Draw a straight line from one point to the other
    With igxBuilder
       .BeginPath
       .MoveTo X1, Y1
       .LineTo X2, Y2
       .EndPath
    End With
    shp.Graphic.Replace igxBuilder.Graphic
    ' Display the result of the Cos and Sin methods
    shp.Text = "Cosine = " & _
```

```
        Round(igxHelper.Cos(igxHelper.Angle(X1, Y1, X2, Y2)), 2) _
        & Chr(13) & "Sine = " & _
        Round(igxHelper.Sin(igxHelper.Angle(X1, Y1, X2, Y2)), 2)
End Sub
```

See Also

Cos method

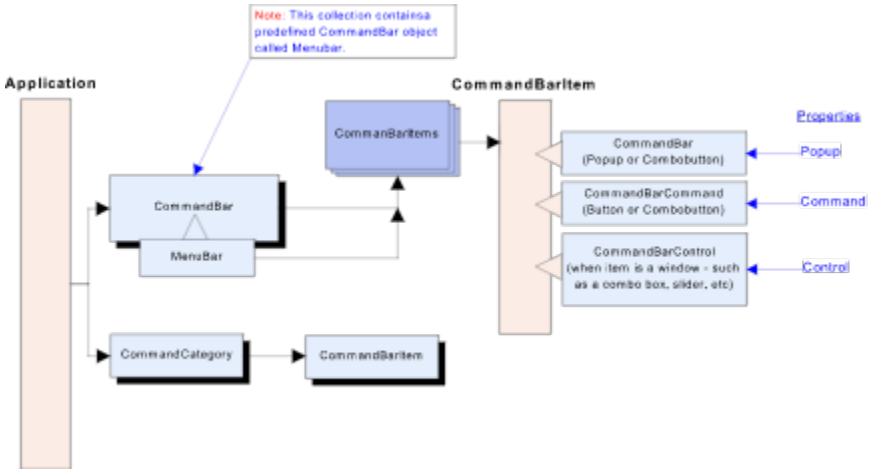
RCos method

RSin method

{button GeometryHelper object,JI(`igrafxrf.HLP',`GeometryHelper_Object')}

CommandBar Object

The CommandBar object represents an individual command bar, such as the Standard Toolbar or the File Menu. A command bar contains one or more buttons; the buttons represent commands that perform some action or activity. The following diagram illustrates the CommandBar object, and some of its properties.



Once you have obtained a CommandBar object, you can navigate further into the Command Bars hierarchy, or you can use the object's methods and properties to modify the command bar.

The example code below illustrates how to get the first CommandBar object from the CommandBars collection.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first CommandBar object from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
```

Properties, Methods, and Events

All of the properties, methods, and events for the CommandBar object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
BuiltIn	Reset	
Caption		
CommandBarItemParent		
CommandBarItems		
Left		
Parent		
Position		
Top		
Visible		

Related Topics

CommandBars object

CommandBarItem object

iGrafx API Object Hierarchy

BuiltIn Property

Syntax *CommandBar*.**BuiltIn**[= {**True | False**}]

Data Type Boolean (read-only)

Description The BuiltIn property indicates whether the specified CommandBar object is a built-in command bar. A built-in command bar is one that shipped with iGrafx Professional.

This property is not valid for a CommandBar object that is derived from a MenuBar object. It always returns True for the CommandBar object in this case.

Example The following example first adds a new, custom CommandBar object called "MyCommandBar" to the application's CommandBars collection. It then iterates through the CommandBars collection and for every built-in CommandBar, it prints the name to the Output window. For every custom Commandbar object, it displayes the name in a message box.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Create a custom CommandBar object
Set igxCmdBar = igxCmdBars.Add
igxCmdBar.Caption = "MyCommandBar"
' Iterate through the CommandBars collection
For Each igxCmdBar In igxCmdBars
    ' List built-in command bars in the Output window
    If (igxCmdBar.BuiltIn) Then
        Output (igxCmdBar.Caption & " is a BuiltIn command bar.")
    Else
        MsgBox (igxCmdBar.Caption & " is not a BuiltIn command bar.")
    End If
Next igxCmdBar
MsgBox "Continue"
```

{button CommandBar object,JI(`igrafxrf.HLP',`CommandBar_Object')}

Caption Property

Syntax *CommandBar*.**Caption**

Data Type String (read/write)

Description The Caption property specifies a caption string (*i.e.* a name) for the CommandBar object. When you add a new, custom command bar to the CommandBars collection, you set the Caption property after you have used the CommandBars.Add method (see the example).

If the command bar is a built-in, you cannot change the value of the Caption property. Attempts to do so generate an error, so you should always check to determine if the CommandBar is a built-in using the BuiltIn property, or write an error handler.

This property is not valid for a CommandBar object that is derived from a MenuBar object.

Example The following example adds a new CommandBar to the collection called "MyCommandBar". It then iterates through the Commandbars collection. If the CommandBar is a built-in, it displays a message indicating that the Caption property cannot be changed. If the CommandBar is not a built-in, then the Caption property is changed to "My Changed Command Bar".

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Create a custom CommandBar object
Set igxCmdBar = igxCmdBars.Add
igxCmdBar.Caption = "MyCommandBar"
' Set up the error handling in case an index
' out of range occurs
On Error GoTo ErrorHandler
' Iterate through the CommandBars collection
For Each igxCmdBar In igxCmdBars
    ' If command bar is not a built-in then change its caption
    If Not (igxCmdBar.BuiltIn) Then
        sOldCaption = igxCmdBar.Caption
        igxCmdBar.Caption = "My Changed Command Bar"
        MsgBox "The Command Bar " & sOldCaption & " has been" _
            & " renamed to " & igxCmdBar.Caption
    Else
        MsgBox (igxCmdBar.Caption & " is a built-in." & _
             "The caption cannot be changed.")
    End If
Next igxCmdBar
' Exit the subroutine
Exit Sub

' This code handles the event of an invalid index
ErrorHandler:
' Display a message box indicating that an invalid
' index was attempted
MsgBox ("Invalid index supplied.")
```

{button CommandBar object,JI(`igrafxrf.HLP',`CommandBar_Object')}

CommandBarItemParent Property

Syntax *CommandBar*.**CommandBarItemParent**

Data Type CommandBarItem object (read-only, See Object Properties)

Description The CommandBarItemParent property returns a CommandBarItem object. When the CommandBar is a CommandBarItem of another CommandBar, this property returns the parent CommandBarItem that would open it.

Example The following example finds the Tools menu CommandBar, and changes the caption of the Tools item on the main command bar.

```
' Dimension the variables
Dim igxToolsMenu As CommandBar
' Find the Tools command bar internally
Set igxToolsMenu = CommandBars.FindBuiltIn(ixToolsMenu)
' Change the caption of the Tools menu parent item
MsgBox "Tools Menu found. Now change the menu item caption."
igxToolsMenu.CommandBarItemParent.Caption = "TOOLS MENU"
' Pause
MsgBox "Click OK to continue."
```

See Also CommandBarItem object

{button CommandBar object,JI(`igrafxrf.HLP',`CommandBar_Object')}

CommandBarItems Property

Syntax *CommandBar*.**CommandBarItems**

Data Type CommandBarItems collection object (read-only, See Object Properties)

Description The CommandBarItems property returns the CommandBarItems collection for the specified CommandBar object (toolbar). A CommandBarItem is a button on the toolbar. Each individual CommandBar object has its own CommandBarItems collection.

Example The following example retrieves the first command bar in the CommandBars collection. It then iterates through all the CommandBarItem objects in the first command bar, and displays the Type property of the CommandBarItem.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
Dim iCount As Integer
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems collection from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
' List the Type property for each CommandBarItem in the collection
For iCount = 1 To igxCmdBarItems.Count
    Select Case igxCmdBarItems.Item(iCount).Type
        Case ixItemButton:
            MsgBox "Item " & iCount & " of " & igxCmdBar.Caption _
                & " is of type Button."
        Case ixItemComboButton:
            MsgBox "Item " & iCount & " of " & igxCmdBar.Caption _
                & " is of type ComboButton."
        Case ixItemControl:
            MsgBox "Item " & iCount & " of " & igxCmdBar.Caption _
                & " is of type Control."
        Case ixItemGroup:
            MsgBox "Item " & iCount & " of " & igxCmdBar.Caption _
                & " is of type Group."
        Case ixItemPopup:
            MsgBox "Item " & iCount & " of " & igxCmdBar.Caption _
                & " is of type Popup."
        Case ixItemSeparator:
            MsgBox "Item " & iCount & " of " & igxCmdBar.Caption _
                & " is of type Separator."
    End Select
Next iCount
```

See Also CommandBarItem object

CommandBarItems object

iGrafx API Object Hierarchy

{button CommandBar object,JI(`igrafxrf.HLP',`CommandBar_Object')}

Left Property

Syntax *CommandBar*.**Left**

Data Type Integer (read/write)

Description The Left property specifies the location of the left side of a CommandBar object (toolbar). The value is specified in pixels. If there is only one command bar at either the Left or Right docking location, changing that command bar's Left property has no effect.

If the Position type is one of the "docked" types, the location of a command bar is relative to the "docking" area on the four edges of the application window. If the Position property is ixFloating, the location is relative to the application window.

Example The following example retrieves the "Standard" command bar from the Commandbars collection, and determines the value of its Position property. Then the command bar's position is changed with the Left and Top properties. Message boxes are used so that the results can be seen. Note that the Application.RefreshUI method is needed so that the UI refreshes while the code is running. Try running this example without using RefreshUI to see what happens.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim sPosition As String
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Determine the setting of the Position property
Select Case igxCmdBar.Position
    Case ixDockTop:
        sPosition = "Docked at Top"
    Case ixDockBottom:
        sPosition = "Docked at Bottom"
    Case ixDockLeft:
        sPosition = "Docked at Left"
    Case ixDockRight:
        sPosition = "Docked at Right"
    Case ixFloating:
        sPosition = "Floating--Not Docked"
End Select
MsgBox "View the position of the " & igxCmdBar.Caption _
    & " command bar." & Chr(13) & "Its Position is " _
    & sPosition
' Set the Left property to 50 pixels, and Top to 100 pixels
igxCmdBar.Left = 50
igxCmdBar.Top = 100
MsgBox "The position of the " & igxCmdBar.Caption _
    & " command bar has been moved 50 pixels to the right," _
    & Chr(13) & "and down 100 pixels." & Chr(13) _
    & "Notice that the Commandbar has disappeared."
' Refresh the UI to make the Command Bar repaint
Application.RefreshUI
MsgBox "The Command Bar has reappeared, but has not been moved" _
    & Chr(13) & "down. This is because there is only one Command" _
    & Chr(13) & "bar docked at the Top."
```

See Also

Position property

Top property

Application.RefreshUI method

{button CommandBar object,JI(`igrafxrf.HLP',`CommandBar_Object')}

Position Property

Syntax *CommandBar*.**Position**

Data Type IxPosition enumerated constant (read/write)

Description The Position property specifies the location at which a command bar is placed in the application's interface. A CommandBar object can be "docked" at either the left, right, top, or bottom of the application window, or it can float anywhere within the window.

If the Position type is one of the "docked" types, the location of a command bar is relative to the "docking" area on the four edges of the application window. If the Position property is ixFloating, the location is relative to the application window.

For menus, this property is always ixBarTop.

The IxPosition constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant	Description
0	ixDockLeft	Command Bar is docked at left of screen
1	ixDockTop	Command Bar is docked at top of screen
2	ixDockRight	Command Bar is docked at right of screen
3	ixDockBottom	Command Bar is docked at bottom of screen
4	ixFloating	Command Bar floats and is not docked

Example The following example gets the 'Standard' command bar from the CommandBars collection, and then displays a message box indicating the command bar's position.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
' Get the CommandBars collection of the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Find("Standard")
' Display the proper message box based on position returned
Select Case igxCmdBar.Position
    Case ixDockBottom
        MsgBox "The command bar is docked on the bottom."
    Case ixFloating
        MsgBox "The command bar is floating."
    Case ixDockLeft
        MsgBox "The command bar is docked on the left."
    Case ixDockRight
        MsgBox "The command bar is docked on the right."
    Case ixDockTop
        MsgBox "The command bar is docked on the top."
End Select
```

See Also Left property

Top property

{button CommandBar object,JI(`igrafxrf.HLP',`CommandBar_Object')}

Reset Method

Topic Under Construction!!! Example not working

Syntax *CommandBar*.**Reset**

Description The Reset method resets a built-in CommandBar object (a toolbar) to its installed defaults. This method only affects the iGrafx Professional built-in command bars; it is not valid for menus (a MenuBar object), or for custom command bars.

[Maybe a note later about this – we'll change it soon so it always works]. This method is equivalent to pressing the Reset button in the Toolbars dialog.

Example The following example shows how to iterate through the CommandBars collection and reset only the built-in toolbars.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
' Get the CommandBars collection of the Application object
Set igxCmdBars = Application.CommandBars
MsgBox "View the default locations of the command bars"
' Find the Standard built-in command bar
Set igxCmdBar = igxCmdBars.Find("Standard")
' Change the command bar's default location
igxCmdBar.Position = ixDockBottom
igxCmdBar.Left = 100
' Find the ToolBox built-in command bar
Set igxCmdBar = igxCmdBars.Find("Toolbox")
' Change the command bar's default location
igxCmdBar.Position = ixFloating
igxCmdBar.Left = 100
igxCmdBar.Top = 100
' Refresh the UI
Application.RefreshUI
MsgBox "View the new locations of the Toolbox command bar"
' Reset the two command bars to their defaults
MsgBox "Click OK to reset the Toolbox command bar to " _
    & "it's default location."
Set igxCmdBar = igxCmdBars.Find("Standard")
igxCmdBar.Reset
Set igxCmdBar = igxCmdBars.Find("Toolbox")
igxCmdBar.Reset
' Refresh the UI
Application.RefreshUI
MsgBox "Default properties restored."
```

{button CommandBar object,JI(`igrafxrf.HLP',`CommandBar_Object')}

Top Property

Syntax *CommandBar*.**Top**

Data Type Integer (read/write)

Description The Top property specifies the location of the top edge of a CommandBar object (toolbar). The value is specified in pixels. If there is only one command bar at either the Top or Bottom docking location, changing that command bar's Top property has no effect.

If the Position type is one of the "docked" types, the location of a command bar is relative to the "docking" area on the four edges of the application window. If the Position property is ixFloating, the location is relative to the application window.

Example The following example retrieves the "Standard" command bar from the CommandBars collection, and determines the value of its Position property. Then the command bar's position is changed with the Left and Top properties. Message boxes are used so that the results can be seen. Note that the Application.RefreshUI method is needed so that the UI refreshes while the code is running. Try running this example without using RefreshUI to see what happens.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim sPosition As String
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Determine the setting of the Position property
Select Case igxCmdBar.Position
    Case ixDockTop:
        sPosition = "Docked at Top"
    Case ixDockBottom:
        sPosition = "Docked at Bottom"
    Case ixDockLeft:
        sPosition = "Docked at Left"
    Case ixDockRight:
        sPosition = "Docked at Right"
    Case ixFloating:
        sPosition = "Floating--Not Docked"
End Select
MsgBox "View the position of the " & igxCmdBar.Caption _
    & " command bar." & Chr(13) & "Its Position is " _
    & sPosition
' Set the Left property to 50 pixels, and Top to 100 pixels
igxCmdBar.Left = 50
igxCmdBar.Top = 100
MsgBox "The position of the " & igxCmdBar.Caption _
    & " command bar has been moved 50 pixels to the right," _
    & Chr(13) & "and down 100 pixels." & Chr(13) _
    & "Notice that the Commandbar has disappeared."
' Refresh the UI to make the Command Bar repaint
Application.RefreshUI
MsgBox "The Command Bar has reappeared, but has not been moved" _
    & Chr(13) & "down. This is because there is only one Command" _
    & Chr(13) & "bar docked at the Top."
```

See Also

Left property

Position property

{button CommandBar object,JI(`igrafxrf.HLP',`CommandBar_Object')}

CommandBars Object

The CommandBars object is a collection of individual CommandBar objects (toolbars). The iGrafx Professional application has only one CommandBars collection. However, this collection can be accessed from either the Application object, the Document object, or the Diagram object. The purpose of the CommandBars collection is to store and provide access to the individual CommandBar objects (toolbars) that have been defined for the application. These include the standard built-in iGrafx Professional toolbars, and any new custom command bars you want to add to the application.

The CommandBars object provides the following functionality:

- The ability to access any CommandBar object that exists within the application.
- The ability to determine how many CommandBar objects are currently in the collection.
- The ability to find (search for) a CommandBar object in the collection based on its Caption property, or if a built-in toolbar, based on its identifier name (an enumerated constant).
- The ability to add a new, custom CommandBar object to the application.
- The ability to set properties that apply to all of the toolbars for the application, such as showing tool tips or using large or small buttons.

The following code shows how to get the Command Bars object from the Application object. The process is the same to get the Command Bars collection from the Document and Diagram objects.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
```

Properties, Methods, and Events

All of the properties, methods, and events for the CommandBars object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
ColorButtons	ExecuteCommand	
Count	Find	
LargeButtons	FindBuiltin	
MenuBar	IsCommandAvailable	
Parent	Item	
ShowToolTips		
WithShortcutKeys		

Related Topics

CommandBar object

iGrafx API Object Hierarchy

Add Method

Syntax *CommandBars*.**Add**([*Caption* As String = "0"]) As CommandBar

Description The Add method adds a new, custom command bar to the CommandBars collection, or a built-in command bar that has been removed from the collection. The new command bar is assigned a generic name (Caption) until you name it (with the CommandBar.Caption property). The new CommandBar becomes the last item in the CommandBars collection. CHANGE THIS BASED ON NEW ARGUMENT.

The method returns a CommandBar object. Acceptable syntax can be either of the following:

```
<CommandBar variable> = igxCmdBars.Add
OR
Call igxCmdBars.Add
```

The first syntactical form is preferred, mostly because you immediately have a CommandBar variable with which you can set the Caption property to give the command bar a name. If you use the second syntactical form, then you must either get the last item in the collection before another command bar is added, use the name assigned by the application, or search for the assigned name and then change the caption.

Example The following example first gets the CommandBars collection from the Application level, the Document level, and the Diagram level to show that all three objects reference the same collection. The Add method is called without assigning the result to a CommandBar variable. This adds a new command bar, but the system generates its own Caption string for it. This is shown by listing the caption of all command bars in the collection to the Output window. The new command bar is located, and the systen assigned name is displayed. Then the new command bar is found based on the system-assigned name, and its caption changed. Last, the captions of all command bars are again displayed in the Output window.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
' Get the CommandBars collection of the Application object
Set igxCmdBars = Application.CommandBars
MsgBox "There are " & igxCmdBars.Count & " command bars at " _
    & "Application level."
Set igxCmdBars = ActiveDocument.CommandBars
MsgBox "There are " & igxCmdBars.Count & " command bars at " _
   & "Document level."
Set igxCmdBars = ActiveDiagram.CommandBars
MsgBox "There are " & igxCmdBars.Count & " command bars at " _
   & "Diagram level."
' Create a custom CommandBar object
Call igxCmdBars.Add
' Display the caption of all command bars in the Output window
For iCount = 1 To igxCmdBars.Count
    Set igxCmdBar = igxCmdBars.Item(iCount)
   Output igxCmdBar.Caption
Next iCount
' Get the last command bar in the collection
Set igxCmdBar = igxCmdBars.Item(igxCmdBars.Count)
' Get the system assigned caption string and display it
sAssignedCaption = igxCmdBar.Caption
MsgBox "The added command bar was named " & sAssignedCaption _
```

```
   & " by the system."
' Set the igxCmdBar variable by finding the assigned caption
Set igxCmdBar = igxCmdBars.Find(sAssignedCaption)
' Assign a new caption to the newly added command bar
igxCmdBar.Caption = "Forgot Name"
' Display the caption of all command bars in the Output window
For iCount = 1 To igxCmdBars.Count
    Set igxCmdBar = igxCmdBars.Item(iCount)
   Output igxCmdBar.Caption
Next iCount
```

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

ColorButtons Property

Syntax *CommandBars*.**ColorButtons**[= {**True | False**}]

Data Type Boolean (read/write)

Description The ColorButtons property specifies whether command bar buttons are displayed with or without color. If the property's value is set to True, the buttons on the command bars are displayed in color; if false, they are displayed without color. This property provides the same functionality as the Color Buttons checkbox in the Toolbars dialog.

Example The following code toggles the color state of the command bar buttons depending on the current color state of the command bars. Message boxes allow you to view the change.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
' Get the CommandBars collection from the Application object.
Set igxCmdBars = Application.CommandBars
' Toggle the color state of the command bars depending
' on current color state
If (igxCmdBars.ColorButtons) Then
    igxCmdBars.ColorButtons = False
Else
    igxCmdBars.ColorButtons = True
End If
MsgBox "View the change to the command bars"
' Toggle the color state of the command bars depending
' on current color state
If (igxCmdBars.ColorButtons) Then
    igxCmdBars.ColorButtons = False
Else
    igxCmdBars.ColorButtons = True
End If
MsgBox "View the change to the command bars"
```

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

ExecuteCommand Method

Syntax *CommandBars*.**ExecuteCommand**(*Command* As IxBuiltInCommand**)**

Description The ExecuteCommand method allows the programmer to execute any of the built in iGrafx Professional commands.

The IxBuiltInCommand constant defines the valid values for the *Command* argument. For the list of values, refer to the table in the Application.ExecuteCommand topic.

Example The following example uses the ExecuteCommand method to open the Components dialog box, and Copy/Paste a diagram.

```
MsgBox "Click OK to open the Components dialog box."
CommandBars.ExecuteCommand (ixFileComponents)
MsgBox "Click OK to Copy the component"
SendKeys "%y", Wait
MsgBox "Click OK to Paste the component"
SendKeys "%P", Wait
MsgBox "Click OK to continue"
```

See Also IsCommandAvailable method

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

Find Method

Syntax *CommandBars*.**Find** (*Caption* As String) As CommandBar

Description The Find method searches the CommandBars collection to find a CommandBar object whose name (Caption property) matches the text string supplied by the *Caption* argument. If no match is found, then the method returns a CommandBar object with with the value "Nothing" in it.

Example The following example retrieves the CommandBars collection and uses the Find method to search for command bar names. It first searches for the built-in command bar named "Standard", which it finds. It then searches for a command bar named "NoName", which it does not find. A message box is displayed indicating success or failure.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
' Get the command bars collection form the Application object
Set igxCmdBars = Application.CommandBars
' Search for the built-in command bar named Standard, and
' set the result to the igxCmdBar variable
sToolBarName = "Standard"
Set igxCmdBar = igxCmdBars.Find(sToolBarName)
' Display the correct message box depending on what is returned
If (igxCmdBar Is Nothing) Then
   MsgBox ("No command bar with the caption, " & sToolBarName _
       & ", was found.")
Else
   MsgBox "Command bar " & igxCmdBar.Caption & " was found."
End If
' Search for a non-existent command bar name
sToolBarName = "NoName"
Set igxCmdBar = igxCmdBars.Find(sToolBarName)
' Display the correct message box depending on what is returned
If (igxCmdBar Is Nothing) Then
   MsgBox ("No command bar with the caption, " & sToolBarName _
       & ", was found.")
Else
   MsgBox "Command bar " & igxCmdBar.Caption & " was found."
End If
```

See Also FindBuiltIn method

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

FindBuiltIn Method

Syntax *CommandBars*.**FindBuiltIn** (*WhichBar* As IxBuiltInCommandBar) As CommandBar

Description The FindBuiltIn method is used to search for one of the pre-defined, built-in command bars that ship with iGrafx Professional. If no match is found, then the method returns a CommandBar object with with the value "Nothing" in it.

The *WhichBar* argument specifies which built-in command bar to find. The argument value must be one of the IxBuiltInCommandBar constants listed in the following table.

Value	Name of Constant
1	ixFileMenu
2	ixEditMenu
3	ixViewMenu
4	ixFormatMenu
5	ixToolsMenu
6	ixArrangeMenu
7	ixWindowMenu
8	ixHelpMenu
9	ixStandardToolbar
10	ixDrawToolbar
11	ixFormattingToolbar
12	ixToolboxToolbar
13	ixPresetsToolbar
14	ixVBAToolbar
15	ixInsertMenu
16	ixCustomDataMenu
17	ixIDiagramMenu
18	ixInsertPictureMenu
19	ixNumberingMenu
20	ixAlignMenu
21	ixMakeSameSizeMenu
22	ixSpaceEvenlyMenu
23	ixGridMenu
24	ixGuidelinesMenu
25	ixRotateFlipMenu
26	ixOrderMenu
27	ixLayersMenu
28	ixConvertToMenu
29	ixCombineMenu

Example The following example retrieves the Commandbars collection, and uses the FindBuiltIn method to search for the "Formatting" command bar. The example also shows what happens if the value of the *WhichBar* argument is not one of the IxBuiltInCommandBar constants. A message box is displayed indicating success or failure.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
```

```
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Set the results of the find to the igxCmdBar variable
Set igxCmdBar = igxCmdBars.FindBuiltIn(ixFormattingToolbar)
' Display the correct message box depending on what is returned
If (igxCmdBar Is Nothing) Then
    MsgBox "Could not find the specified command bar."
Else
    MsgBox "Command bar named, " & igxCmdBar.Caption & ", was found."
End If
' Search for a non-existent command bar name
Set igxCmdBar = igxCmdBars.FindBuiltIn(ixBogusName)
' Display the correct message box depending on what is returned
If (igxCmdBar Is Nothing) Then
    MsgBox "Could not find the specified command bar."
Else
    MsgBox "Command bar named, " & igxCmdBar.Caption & ", was found."
End If
```

See Also Find method

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

IsCommandAvailable Method

Syntax *CommandBars*.**IsCommandAvailable**(*Command* As IxBuiltInCommand) As Boolean

Description The IsCommandAvailable method lets you determine whether a built-in command is enabled or available in the user interface. For example, the command Edit—Copy command (ixEditCopy) is not available when there is no selection in the active diagram, or there is no active diagram. Note that the ExecuteCommand method does not let you execute commands that are not available.

For the list of valid values for the IxBuiltInCommand constant, refer to the Application.ExecuteCommand method.

Example The following example executes the Copy item, but only if it's available. If not, it reports the result.

```
' Execute the Copy item if available
If CommandBars.IsCommandAvailable(ixEditCopy) Then
   ActiveDiagram.Copy
    MsgBox "Copy successful"
Else
   MsgBox "Copy item not available. Nothing selected to copy."
End If
```

See Also ExecuteCommand method

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

Item Method

Syntax *CommandBars*.**Item**(*Index*) As CommandBar

Description The Item method returns the CommandBar object at the specified *Index* from the CommandBars collection.

The *Item* argument is of type Variant. It accepts either a number or a string. If you supply a number, it returns CommandBar object based on it's index in the collection. If you supply a string, it searches all the CommandBar Caption properties for a matching string, and returns the CommandBar object, if found.

Error If you supply an invalid index number, or a string that doesn't exist, then an Invalid Index Value error is returned. Use error trapping if your code could potentially supply the *Index* argument with an invalid value.

Example The following example uses the Item method to list all the CommandBars in the collection, along with their index numbers.

```
' List all the Commandbars in the system, and their Index numbers
For Index = 1 To CommandBars.Count
    sString = sString & Str(Index) & " - " & _
    CommandBars.Item(Index).Caption & Chr(13)
Next Index
MsgBox "All current Commandbars:" & Chr(13) & Chr(13) & sString
```

You can also refer to the example for the Add method to see the use of the Item method.

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

LargeButtons Property

Syntax *CommandBars*.**LargeButtons**[= {**True | False**}]

Data Type Boolean (read/write)

Description The LargeButtons property specifies whether to use large or small buttons on the command bars. The property affects all CommandBar objects in the collection. The property provides the same functionality as setting the Large Buttons checkbox in the Toolbars dialog.

Example The following code toggles the "large buttons" state of the command bar buttons depending on the current "large buttons" state. Message boxes allow you to view the change.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Toggle the LargeButtons state of the command bars depending
' on current LargeButtons state
If (igxCmdBars.LargeButtons) Then
    igxCmdBars.LargeButtons = False
Else
    igxCmdBars.LargeButtons = True
End If
MsgBox "View the change to the command bars"
' Toggle the LargeButtons state of the command bars depending
' on current LargeButtons state
If (igxCmdBars.LargeButtons) Then
    igxCmdBars.LargeButtons = False
Else
    igxCmdBars.LargeButtons = True
End If
MsgBox "View the change to the command bars"
```

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

MenuBar Property

Syntax *CommandBars*.**MenuBar** As CommandBar

Data Type CommandBar object (read-only, See Object Properties)

Description The MenuBar property returns the CommandBar object from the CommandBars collection that is specifically the application's menu bar. The MenuBar is a specific CommandBar. There is only one for the application, and it can be accessed only through this property (see the example).

Most of the properties and methods of a CommandBar object do not work for the menu bar, such as:

- The Position, Left, Top, and Visible properties have no effect.
- The BuiltIn property always returns True.
- The Delete method has no effect.

Example The following example shows how to add a command to the main MenuBar. In this case, the Undo and Redo commands are added. This might be useful if you often use the Undo and Redo commands.

```
' Dimension the variables
Dim igxMenuBar As CommandBar
' Get the MenuBar object
Set igxMenuBar = CommandBars.MenuBar
' Add the Undo command to the MenuBar
igxMenuBar.CommandBarItems.AddBuiltIn (ixEditUndo)
igxMenuBar.CommandBarItems.AddBuiltIn (ixEditRedo)
If MsgBox("Undo/Redo added to the main Menu Bar." _
& " Remove these items?", vbYesNo) = vbYes Then
    igxMenuBar.Reset
End If
```

See Also CommandBar object

iGrafx API Object Hierarchy

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

ShowTooltips Property

Syntax *CommandBars*.**ShowTooltips**[= {**True | False**}]

Data Type Boolean (read/write)

Description The ShowToolTips property specifies whether ToolTips text is displayed on the command bars. A ToolTip is text that pops up when the cursor is placed over a toolbar bar button, telling the user what the button does. The property affects all CommandBar objects in the collection. The property provides the same functionality as setting the Show ToolTips checkbox in the Toolbars dialog.

Example The following example toggles the state of the ShowTooltips property depending on its current state. Test whether tool tips are being displayed before you run the code.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
' Get the command bars collection form the Application object
Set igxCmdBars = Application.CommandBars
' Toggle the ToolTip state of the command bars depending on
' current ToolTip state
If (igxCmdBars.ShowTooltips) Then
    igxCmdBars.ShowTooltips = False
Else
    igxCmdBars.ShowTooltips = True
End If
MsgBox "Click OK to end the subroutine. Go to the diagram" _
    & Chr(13) & "window and hold the cursor over a toolbar button."
```

See Also WithShortcutKeys property

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

WithShortcutKeys Property

Syntax *CommandBars*.**WithShortcutKeys**[= {**True | False**}]

Data Type Boolean (read/write)

Description The WithShortcutKeys property specifies whether shortcut keys are displayed with the ToolTips for the command bars. Setting this value is the same as setting the With Shortcut Keys option in the Toolbars dialog.

Example The following code toggles the state of the WithShortcutKeys property depending on its current state. You may want to check whether shortcut keys are included in the tool tips before running this code.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Toggle the ToolTip state of the command bars depending
' on current ToolTip state
If (igxCmdBars.WithShortcutKeys) Then
   igxCmdBars.WithShortcutKeys = False
Else
   igxCmdBars.WithShortcutKeys = True
End If
MsgBox "Click OK to end the subroutine. Go to the diagram" _
    & Chr(13) & "window and hold the cursor over a toolbar button."
```

See Also ShowTooltips property

{button CommandBars object,JI(`igrafxrf.HLP',`CommandBars_Object')}

CommandBarCommand Object

The CommandBarCommand object specifies the command (which is going to perform some action or activity) that is associated with a toolbar button. For instance, the Save command is associated with the Save button on the "Standard" toolbar, and the File—Save menu option.

Using the Execute method of this object is equivalent to selecting a menu option or clicking a toolbar button through the user interface.

This object is accessible only from the CommandBarItem object's Command property.

Only a command bar item of type Button has a command associated with it. Therefore, trying to access this object from one of the other types of command bar items returns a "Not a Command" error.

Properties, Methods, and Events

All of the properties, methods, and events for the CommandBarCommand object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Execute	
Enabled		
ID		
Parent		
State		

Related Topics

CommandBarItem object

Enabled Property

Syntax *CommandBarCommand*.**Enabled**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Enabled property specifies whether a custom CommandBarItem is enabled. If set to True, the CommandBarItem appears active in the menu or toolbar. If set to False, the CommandBarItem appears inactive ("grayed out"), and it's command will not execute. This property has no effect on built-in command bar items. For this reason, a command bar item should be tested to see if it is a built-in item.

The Enabled property must be set inside the Update Event of a CommandHandler Class you have written. If you set this property outside of a CommandHandler Class it has no effect.

Example The following example creates a new item on the Edit menu called "Make Blue". This command changes the fill color to blue of all selected shapes. The Enabled property is used to make the CommandBarItem enabled only when one or more objects have been selected.

The following code is the CommandHandler Class. Copy these routines into a new Class Module called Class1. Insert a new Class Module into your project by using the Insert->Class Module menu item in the Visual Basic editor.

```
Implements CommandHandler
' "Implements CommandHandler" makes this Class a CommandHandler.
' You can then use it to create new CommandHandler objects
' that use your custom code. It also provides 3 default events.

Private Sub CommandHandler_Execute()
   ' Change all the shapes in the selection blue
   For Index = 1 To ActiveDiagram.Selection.Count
      If ActiveDiagram.Selection.Item(Index).Type _
      = ixObjectShape Then
          ActiveDiagram.Selection.Item(Index).Shape.FillColor = vbBlue
      End If
   Next Index
End Sub

Private Sub CommandHandler_Help()
End Sub

Private Sub CommandHandler_Update(ByVal Command As _
IXCommandBarCommand)
   ' Enable the menu item only if something is selected
   If ActiveDiagram.Selection.Count > 0 Then
       Command.Enabled = True
   Else
       Command.Enabled = False
   End If
End Sub
```

The following code is the Main( ) program. Copy this block of code into your project's Diagram code pane. Run the Main( ) subroutine to set up the new menu item.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxHandler As New Class1
```

```
    Dim igxMyItem As CommandBarItem
    Dim igxComBarCommand As CommandBarCommand
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    ' Add a new menu item to the Edit menu
    Set igxMyItem = CommandBars.FindBuiltIn(ixEditMenu) _
        .CommandBarItems.AddButton("Make Blue", igxHandler)
    ' Add two shapes to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 4)
    ' Pause
    MsgBox "Try selecting shapes, then use the Edit->Make Blue item"
End Sub
```

{button CommandBarCommand object,JI(`igrafxrf.HLP',`CommandBarCommand_Object')}

Execute Method

Syntax *CommandBarCommand*.**Execute**

Description The Execute method is used to execute a command without clicking on the command bar item (a button through the interface). Using this method is equivalent to clicking a CommandBarItem with the mouse.

If the CommandBarItem is a custom item using a CommandHandler you have written, the Execute method fires the Execute event of your CommandHandler Class object.

Example The following example uses the Execute method to open the Components dialog box.

```
' Dimension the variables
Dim igxMenu As CommandBar
Dim igxItem As CommandBarItem
Dim igxCommand As CommandBarCommand
' Get the menu object
Set igxMenu = Application.CommandBars.FindBuiltIn(ixFileMenu)
' Get the menu item object
Set igxItem =  igxMenu.CommandBarItems.FindBuiltInItem(ixFileComponents)
' Get the item's command object
Set igxCommand = igxItem.Command
' Execute the command
MsgBox "Click OK to open the Components dialog box"
igxCommand.Execute
MsgBox "Click OK to continue"
```

See Also CommandHandler object

{button CommandBarCommand object,JI(`igrafxrf.HLP',`CommandBarCommand_Object')}

ID Property

Syntax *CommandBarCommand*.**ID**

Data Type Long (read-only)

Description The ID property returns the control ID of the specified CommandBarCommand object. All existing CommandBarCommands have an ID, and new Commands are assigned an ID when they are created.

Example The following example lists the ID's of all the CommandBarCommands on the File Menu.

```
'Dimension the variables
Dim strText As String
' Gather IDs and Captions for all items in the File Menu
With CommandBars.FindBuiltIn(ixFileMenu).CommandBarItems
   For Index = 1 To .Count
      If .Item(Index).Type = ixItemButton Then
         strText = strText & .Item(Index).Command.ID & " - " _
            & .Item(Index).Caption & Chr(13)
      End If
   Next Index
End With
' Display the result
MsgBox "File Menu Items:" & Chr(13) & _
    "ID        Caption" & Chr(13) & strText
```

{button CommandBarCommand object,JI(`igrafxrf.HLP',`CommandBarCommand_Object')}

State Property

Syntax *CommandBarCommand*.**State**

Data Type IxCommandState enumerated constant (read/write)

Description The State property specifies the operating state of a command bar item. It allows the developer to set and return the checked state of a command. If the command is a CheckBox, the State is Checked or NotChecked. If the command is a RadioButton the State is RadioCheck or NotChecked.

When a command bar item is in a menu, the State property is used to make an item checked or unchecked. When a command bar item is a button, the property makes the button pressed or not pressed.

The IxCommandState constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant
0	ixCommandNotChecked
1	ixCommandChecked
2	ixCommandRadioCheck

Example The following displays and changes the checked state of the Gallery Menu Item in the View Menu. This item is a CheckBox menu item.

```
Private Sub Main()
   ' Dimension variable
   Dim igxItem As CommandBarItem
   Set igxItem = CommandBars.FindBuiltIn(ixViewMenu) _
       .CommandBarItems.Item(10)
   ' Report the state of the Gallery Menu Item
   MsgBox "Gallery Menu Item is " & StateString(igxItem.Command)
   ' Uncheck the command's state
   MsgBox "Click OK to uncheck the Gallery Menu Item"
   igxItem.Command.State = ixCommandNotChecked
   ' Report the state
   MsgBox "Gallery Menu Item is " & StateString(igxItem.Command)
   igxItem.Command.State = ixCommandChecked
   MsgBox "Click OK to continue"
End Sub

' Return a string for the state of a command
Private Function StateString(Command As CommandBarCommand) As String
   Select Case Command.State
       Case 0
           StateString = "not checked"
       Case 1
           StateString = "checked"
       Case 2
           StateString = "radio checked"
   End Select
End Function
```

{button CommandBarCommand object,JI(`igrafxrf.HLP',`CommandBarCommand_Object')}

CommandBarControl Object

The CommandBarControl object represents a VB control object that can be used as a CommandBarItem object. This object is a placeholder for future development. At present, there are no properties that access this object.

Properties, Methods, and Events

All of the properties, methods, and events for the CommandBarControl object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Parent		

Related Topics

CommandBarItem object

CommandBarItem Object

The CommandBarItem object represents an individual toolbar button on a command bar, or a menu item on a menu bar. This object is subordinate to the CommandBar object (a toolbar or menu).

iGrafx Professional contains a large collection of built in CommandBarItems, and you can add your own custom items to expand the features accessed from menus and toolbars. Built-in items can be modified as well as custom items. CommandBarItems allow you to move them to other command bars, make them invisible, change the caption and description text, change the icon image, and more.

Properties, Methods, and Events

All of the properties, methods, and events for the CommandBarItem object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
BuiltIn		
Caption		
Command		
Control		
DescriptionText		
Image		
Index		
Parent		
Popup		
Style		
Type		
Visible		

Related Topics

CommandBarItems object
CommandBarCommand object
CommandBarControl object
CommandBarItemGroup object
iGrafx API Object Hierarchy

BuiltIn Property

Syntax *CommandBarItem*.**Builtin**[= {**True | False**}]

Data Type Boolean (read-only)

Description The Builtin property indicates whether the specified CommandBarItem object is a built-in command bar item. A built-in command bar item is one that shipped with iGrafx Professional. Custom command bar items added by developers are not built-ins, and would return a value of False.

Example The following example displays a message box if the first command bar item in the command bars collection is a built-in. The example also shows that you cannot change the Caption property of a built-in command bar item.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
Dim igxCmdBarItem As CommandBarItem
Dim sOldName As String
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems object from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
' Get the first CommandBar item from the CommandBarItems
' collection
Set igxCmdBarItem = igxCmdBarItems.Item(1)
' Display a message box if the CommandBar item is a built-in
If (igxCmdBarItem.BuiltIn) Then
    MsgBox (igxCmdBarItem.Caption & " is a built-in.")
End If
sOldName = igxCmdBarItem.Caption
igxCmdBarItem.Caption = "My New"
MsgBox igxCmdBarItem.Caption & " is a built-in."
igxCmdBarItem.Caption = sOldName
MsgBox igxCmdBarItem.Caption & " is a built-in."
```

{button CommandBarItem object,JI(`igrafxrf.HLP',`CommandBarItem_Object')}

Command Property

Topic Under Construction!!! Test the example

Syntax *CommandBarItem*.**Command**

Data Type CommandBarCommand object (read-only, See Object Properties)

Description The Command property returns the CommandBarCommand object that is associated with the specified CommandBarItem object (a toolbar button).

Before attempting to get the CommandBarCommand object, you should first test the command bar item to see if is a button (type *ixItemButton*). Only buttons have an associated Command object.

Example The following code gets the CommandBarCommand object of a command bar item. If it is the type of item that would have a CommandBarCommand object associated with it, the proper message is then output to the Output window.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
Dim igxCmdBarCmd As CommandBarCommand
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the Command Bars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems object from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
' Get the Save command bar item from the command bar
For Each Item In igxCmdBarItems
' Display the proper message box based on the CommandBarItem type
    Select Case Item.Type
       Case ixItemPopup
           Output Item.Caption & " does not have a" & _
               " command object."
       Case ixItemButton
           ' If item is a button, get the command object
           Set igxCmdBarCmd = Item.Command
           ' Output the command ID of the command to
           ' the Immediate window
           Output Item.Caption & " has a command ID" & _
               " of " & igxCmdBarCmd.ID
       Case ixItemSeparator
           Output "A Separator does not have a" & _
               " command object."
       Case ixItemControl
           Output Item.Caption & " does not have a" & _
               " command object."
       Case ixItemComboButton
           Output Item.Caption & " does not have a" & _
               " command object."
       Case ixItemGroup
           Output Item.Caption & " does not have a" & _
```

```
                    " command object."
        End Select
    Next
```

See Also CommandBarCommand object

iGrafx API Object Hierarchy

{button CommandBarItem object,JI(`igrafxrf.HLP',`CommandBarItem_Object')}

Control Property

Syntax *CommandBarItem*.**Control**

Data Type CommandBarControl object (read-only, See Object Properties)

Description The Control property returns the CommandBarControl object that is associated with the specified CommandBarItem object (a toolbar button). The CommandBarControl object represents the control that is associated with the CommandBarItem. Not all command bar items have control objects associated with them. For this reason, the command bar item should be tested before retrieving the CommandBarControl object.

Important Currently this property is a place holder for future expansion of the iGrafx Professional API, and does not yet implement any properties or methods.

Example The following example displays the caption of the parent of a control to the Immediate window, or a message indicating that the command bar item does not have a CommandBarControl object associated with it.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
Dim igxCmdBarControl As CommandBarControl
' Get Command Bars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the Command Bars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems object from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
' Go through all command bar items in the command bar and
' display appropriate messages
For Each Item In igxCmdBarItems
    ' Display the proper message box based on the CommandBarItem type
   Select Case Item.Type
       Case ixItemPopup
           Output Item.Caption & " does not have a" & _
               " control object."
       Case ixItemButton
           Output Item.Caption & " does not have a" & _
               " control object."
       Case ixItemSeparator
           Output "A separator does not have a" & _
               " control object."
       Case ixItemControl
           ' If the item is a control, then get
           ' the Control object
           Set igxCmdBarControl = Item.Control
           ' Output the caption of the Control object's
           ' parent to the Immediate window
           Output Item.Caption & "'s parent is " & _
               igxCmdBarControl.Parent.Caption
       Case ixItemComboButton
           Output Item.Caption & " does not have a" & _
               " control object."
       Case ixItemGroup
           Output Item.Caption & " does not have a" & _
```

```
                " control object."
    End Select
Next
```

See Also CommandBarControl object

iGrafx API Object Hierarchy

{button CommandBarItem object,JI(`igrafxrf.HLP',`CommandBarItem_Object')}

DescriptionText Property

Syntax *CommandBarItem*.**DescriptionText**

Data Type String (read/write)

Description The DescriptionText property specifies the text that is displayed as a hint for the specified CommandBarItem object, if it is a command. You use this property for your own custom commands; it has no effect (and returns a error) if the command bar item is a built-in. For this reason, you should check to see if a toolbar button (CommandBarItem) is a built-in before changing the DescriptionText property.

Example The following example gets the first command bar item from the first command bar, and if it is not a built-in, changes its description text.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
Dim igxCmdBarItem As CommandBarItem
' Get the command bars collection form the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems object from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
' Get the first command bar item from the command bar
Set igxCmdBarItem = igxCmdBarItems.Item(1)
' Check to see if it is a built-in command bar
' If it is not then change the description text
If Not (igxCmdBarItem.BuiltIn) Then
    ' Set the description text of the command bar item
    igxCmdBarItem.DescriptionText = "My Command Bar Item"
End If
```

{button CommandBarItem object,JI(`igrafxrf.HLP',`CommandBarItem_Object')}

Image Property

Topic Under Construction!!! Example not working

Syntax *CommandBarItem*.**Image**

Data Type StdPicture object (read-only, See Object Properties)

Description The Image property returns a Visual Basic StdPicture object. Most CommandBarItems on a toolbar are displayed as a square button with an image. Menu items are displayed as a small image next to a caption in the menu. The Image property returns the Image as a StdPicture in either case.

Example The following example gets the image from a toobar item and displays it's width and height.

```
' Dimension the variables
Dim igxBar As CommandBar
Dim igxItem As CommandBarItem
Dim igxPicture As StdPicture
' Get the Standar Toolbar object
Set igxBar = CommandBars.FindBuiltIn(ixStandardToolbar)
' Get the Shape Palete item object
Set igxItem = igxBar.CommandBarItems.Item(17)
' Get the picture
Set igxPicture = igxItem.Image
MsgBox "The Shape Palette button has an image " & _
    igxPicture.Height & " x " & igxPicture.Width
```

{button CommandBarItem object,JI(`igrafxrf.HLP',`CommandBarItem_Object')}

Index Property

Syntax *CommandBarItem*.**Index**

Data Type Integer (read-only)

Description The Index property returns the index value of the specified CommandBarItem object (a toolbar button or separator). The index value is the position of the command bar item on the command bar (that is, its order in the command bar, and therefore, its position in the CommandBarItems collection).

A separator is also a command bar item, so it too has an index value. Remember to take separators into account when determining the index of a command bar item visually.

Example The following example gets the index of a command bar item and displays it in the Output window.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
Dim igxCmdBarItem As CommandBarItem
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems object from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
' Get the Save command bar item from the command bar
Set igxCmdBarItem = igxCmdBarItems.Find("Save")
' Output the index value of the command bar item to the
' Output window
Output "Index = " & igxCmdBarItem.Index
```

Popup Property

Syntax *CommandBarItem*.**Popup**

Data Type CommandBar object (read-only, See Object Properties)

Description The Popup property returns a CommandBar object for the specified CommandBarItem object. A popup command bar is a menu accessed from another menu, in which case the menu is a CommandBarItem of another menu. The Popup property returns the menu CommandBar object for the sub menu.

This property is valid only if the CommandBarItem is a popup (Type property equals ixItemPopup). In this case, the CommandBar object is used to define or access the items that appear on the popup (see the Type property). If the command bar item is not a popup, then the value 'Nothing' is returned.

Example The following example uses command bars and command bar items to create a new Document and Process diagram using the File->New->Process menu item. The New menu is a sub menu of the File menu—a popup.

```
' Dimension the variables
Dim igxFileMenu As CommandBar
Dim igxNewItem As CommandBarItem
Dim igxNewMenu As CommandBar
Dim igxProcessItem As CommandBarItem
' Get the File Menu object
Set igxFileMenu = CommandBars.FindBuiltIn(ixFileMenu)
' Get the New menu item object
Set igxNewItem = igxFileMenu.CommandBarItems.Item(1)
' Get the menu object under the New item
Set igxNewMenu = igxNewItem.Popup
' Get the Process item in the New menu
Set igxProcessItem = igxNewMenu.CommandBarItems.Item(1)
' Execute the menu item
MsgBox "Click OK to invoke the File->New->Process menu item."
igxProcessItem.Command.Execute
```

See Also CommandBar object

iGrafx API Object Hierarchy

{button CommandBarItem object,JI(`igrafxrf.HLP',`CommandBarItem_Object')}

Style Property

Syntax *CommandBarItem*.**Style**

Data Type IxCommandBarItemStyle enumerated constant (read/write)

Description The Style property specifies the display style to use for the specified CommandBarItem object—a toolbar button, or menu item.

Toolbar buttons are usually displayed with image only, no caption (Style 2). Menu items are usually displayed with image and caption (Style 3)

The IxCommandBarItemStyle constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
1	ixCommandBarItemCaption
2	ixCommandBarItemImage
3	ixCommandBarItemImageAndCaption

Example The following example changes the style of all the items in the Standard Toolbar to captions only, then back to images only.

```
' Dimension the variables
Dim igxStandardToolbar As CommandBar
' Get the Standard Toolbar object
Set igxStandardToolbar = CommandBars.FindBuiltIn(ixStandardToolbar)
' Work with it's command bar items
With igxStandardToolbar.CommandBarItems
    MsgBox "Click OK to make the Standard Toolbar all captions."
    ' Set all the items to captions
    For Index = 1 To .Count
       If Not .Item(Index).Type = ixItemSeparator Then
          .Item(Index).Style = ixCommandBarItemCaption
       End If
    Next Index
    MsgBox "Click OK to set them back to images."
    ' Set all the items back to images
    For Index = 1 To .Count
       If Not .Item(Index).Type = ixItemSeparator Then
          .Item(Index).Style = ixCommandBarItemImage
       End If
    Next Index
End With
MsgBox "Click OK to continue"
```

{button CommandBarItem object,JI(`igrafxrf.HLP',`CommandBarItem_Object')}

Type Property

Syntax *CommandBarItem*.**Type**

Data Type IxCommandBarItemType enumerated constant (read-only)

Description The Type property returns the type of the specified CommandBarItem. A CommandBarItem of any of the types listed in the table are added to a CommandBar using the methods of the CommandBarItems object.

The IxCommandBarItemType constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixItemPopup
1	ixItemButton
2	ixItemSeparator
3	ixItemControl
4	ixItemComboButton
5	ixItemGroup

Example The following example iterates through the first command bar and displays a message in the Output window indicating the type of each CommandBarItem.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the Command Bars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems object from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
' Get the Save command bar item from the command bar
For Each Item In igxCmdBarItems
    ' Display message in Output window based on CommandBarItem type
    Select Case Item.Type
       Case ixItemPopup
           Output Item.Caption & " is a PopUp."
       Case ixItemButton
           Output Item.Caption & " is a Button."
       Case ixItemSeparator
           Output Item.Caption & " is a Separator."
       Case ixItemControl
           Output Item.Caption & " is a Control."
       Case ixItemComboButton
           Output Item.Caption & " is a ComboButton."
       Case ixItemGroup
           Output Item.Caption & " is a Group."
    End Select
Next
```

{button CommandBarItem object,JI(`igrafxrf.HLP',`CommandBarItem_Object')}

CommandBarItemGroup Object

The CommandBarItemGroup object is a group of CommandBarItem objects. This object is a placeholder for future development. At present, there are no properties that access this object.

Properties, Methods, and Events

All of the properties, methods, and events for the CommandBarItemGroup object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Parent		

Related Topics

CommandBarItem object

CommandBarItems Object

The CommandBarItems object is a collection of individual CommandBarItem objects (toolbar buttons or menu items). Each CommandBar object has its own CommandBarItems collection. Its purpose is to store and provide access to the individual CommandBarItem objects (toolbar buttons and other controls) that have been defined for a specific command bar.

The CommandBarItems object provides the following functionality:

- The ability to access any CommandBarItem objects that have been created for a specific CommandBar object.
- The ability to determine how many CommandBarItem objects are currently in the collection.
- The ability to find (search for) specific types of CommandBarItem objects that may exist in the collection.
- The ability to add a new CommandBarItem object of a specific type to a CommandBar object.

The following code example illustrates how to get the CommandBarItems collection from the first command bar in the CommandBars collection.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
' Get the CommandBars collection form the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems object from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
```

Properties, Methods, and Events

All of the properties, methods, and events for the CommandBarItems object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	AddBuiltIn	
Parent	AddButton	
	AddPopup	
	AddSeparator	
	Find	
	FindBuiltinItem	
	Item	

Related Topics

CommandBarItem object

iGrafx API Object Hierarchy

Add Method

Syntax *CommandBarItems*.**Add (***CommandBarItemToInsert* As CommandBarItem, *Index* As Integer) As CommandBarItem

Description The Add method adds a CommandBarItem object to the specified CommandBarItems collection by copying it from another CommandBar or from a CommandCategory. The result of the method must be assigned to a variable of type CommandBarItem.

The *CommandBarItemToInsert* argument specifies the item you want to add to the collection. The *Index* argument specifies the location within the collection where you want to insert the added CommandBarItem. If *Index* is 0, the CommandBarItem is added to the end of the command bar.

Example The following example gets the first command bar item, the Font drop down list, from the second command bar, and adds it to the beginning of the first command bar.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar1 As CommandBar
Dim igxCmdBar2 As CommandBar
Dim igxCmdBarItems As CommandBarItems
Dim igxCmdBarItem As CommandBarItem
Dim igxTest As CommandBarItem
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar1 = igxCmdBars.Item(1)
' Get the second command bar from the Command Bars collection
Set igxCmdBar2 = igxCmdBars.Item(2)
' Get CommandBarItems collection from the 2nd CommandBar object
Set igxCmdBarItems = igxCmdBar2.CommandBarItems
' Get the first item from the second command bar, which is
' the font drop down
Set igxCmdBarItem = igxCmdBarItems.Item(1)
' Add the item to the first command bar as the first item
Set igxTest = igxCmdBar1.CommandBarItems.Add _
    (igxCmdBarItem, 1)
```

{button CommandBarItems object,JI(`igrafxrf.HLP',`CommandBarItems_Object')}

AddBuiltIn Method

Syntax *CommandBarItems*.**AddBuiltIn**(*BuiltInItem* As IxBuiltInCommand, [*Index* As Integer]) As CommandBarItem

Description The AddBuiltIn method adds one of the iGrafx Professional built-in commands to the specified CommandBarItems collection. The result of the method must be assigned to a variable of type CommandBarItem.

The *BuiltInItem* argument specifies the built-in command to add. The IxBuiltInCommand constant defines the valid values (see the table in the discussion of the Application.ExecuteCommand method). The *Index* argument specifies the location within the collection where you want to insert the built-in command. If *Index* is not specified, the built-in command is added to the end of the command bar.

Example The following example adds the Connect Shapes command to the Standard Toolbar.

```
' Dimension the variables
Dim igxStandard As CommandBar
' Get the Menu Bar object
Set igxStandard = CommandBars.FindBuiltIn(ixStandardToolbar)
MsgBox "Click OK to add the ConnectShapes command to " _
   & "the Standard Toolbar"
' Add the ConnectShapes command to the standard toolbar
igxStandard.CommandBarItems.AddBuiltIn (ixConnectShapes)
MsgBox "Click OK to continue"
```

See Also CommandBarItem object

{button CommandBarItems object,JI(`igrafxrf.HLP',`CommandBarItems_Object')}

AddButton Method

Syntax *CommandBarItems*.**AddButton**(*Name* As String, *Handler* As CommandHandler, [*Index* As Integer]) As CommandBarItem

Description The AddButton method adds a new custom command to the CommandBarItems collection. The AddButton method allows you to add a custom commands that you write yourself. You can add custom commands to menus and toolbars.

The *Name* argument specifies the name of the command item. This string becomes the caption for the command initially. The caption can be changed later.

The *Handler* argument specifies a CommandHandler object that executes your code when the command is used. The CommandHandler object is derived from a CommandHandler class that you have written (see the example below.)

The *Index* argument specifies the position to insert the new custom command. If you do not specify a value, the command is inserted at the end of the collection—the bottom of a menu, or the left side of a tool bar.

The Type property for the item added with this method is set to *ixItemButton*.

The AddButton method requires a command handler. If you need to understand the CommandHandler object, refer to the CommandHandler object, the CommandBarCommand.Enabled property, or the example below.

Example The following example creates a new item on the Edit menu called "Make Blue". It will change the fill color to blue on any selected shapes. The Enabled property is used to make the item enabled only when one or more objects have been selected with the mouse.

The following code is the CommandHandler Class. Copy these routines into a new Class Module called Class1. Insert a new Class Module into your project by using the Insert->Class Module menu item in the Visual Basic editor.

```
Implements CommandHandler
' "Implements CommandHandler" makes this Class a CommandHandler.
' You can then use it to create new CommandHandler objects
' that use your custom code. It also provides 3 default events.

Private Sub CommandHandler_Execute()
    ' Change all the shapes in the selection blue
    For Index = 1 To ActiveDiagram.Selection.Count
        If ActiveDiagram.Selection.Item(Index).Type _
        = ixObjectShape Then
            ActiveDiagram.Selection.Item(Index).Shape.FillColor _
            = vbBlue
        End If
    Next Index
End Sub

Private Sub CommandHandler_Help()
End Sub

Private Sub CommandHandler_Update(ByVal Command As _
IXCommandBarCommand)
    ' Enable our menu item only if something is selected
    If ActiveDiagram.Selection.Count > 0 Then
        Command.Enabled = True
    Else
        Command.Enabled = False
```

```
    End If
End Sub
```

The following code is the Main( ) program. Copy this block of code into your project's Diagram code pane. Run the Main( ) subroutine to set up the new menu item.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxHandler As New Class1
    Dim igxMyItem As CommandBarItem
    Dim igxComBarCommand As CommandBarCommand
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    ' Add a new menu item to the Edit menu
    Set igxMyItem = CommandBars.FindBuiltIn(ixEditMenu) _
       .CommandBarItems.AddButton("Make Blue", igxHandler)
    ' Add two shapes to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 2)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
       (1440 * 2, 1440 * 4)
    ' Pause
    MsgBox "Try selecting shapes, then use the Edit->Make Blue item"
End Sub
```

See Also CommandBarItem object

CommandHandler object

{button CommandBarItems object,JI(`igrafxrf.HLP',`CommandBarItems_Object')}

AddPopup Method

Syntax *CommandBarItems*.**AddPopup**([*CmdBar* As CommandBar], [*Index* As Integer]) As CommandBarItem

Description The AddPopup method adds an existing CommandBar object to a command bar as a popup. The method returns the new command bar item. A Popup CommandBar is another menu. The added popup will be a sub menu of the CommandBar.

The *CmdBar* argument specifies an existing CommandBar object to add. If you omit this argument, an empty popup CommandBar is added. It can be filled in later with CommandBarItems (see the example below).

The *Index* argument specifies the location in the command bar to add the command bar. If you omit this argument, the new popup will be added to the end of the collection—the bottom of a menu, or the left side of a toolbar.

Example The following example creates a new menu called "Zoom". It then adds all the available zoom commands to the new menu.

```
' Dimension the variables
Dim igxMenuBar As CommandBar
Dim igxPopup As CommandBarItem
' Get the MenuBar object
Set igxMenuBar = CommandBars.MenuBar
' Add a new menu to the MenuBar
Set igxPopup = igxMenuBar.CommandBarItems.AddPopup()
' Call the new menu "Zoom"
igxPopup.Caption = "Zoom"
' Add all the zoom tools to our new menu
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomComboBox)
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomIn)
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomOut)
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomPrevious)
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomTool)
MsgBox "New Zoom menu is ready."
```

See Also CommandBar object

CommandBarItem object

{button CommandBarItems object,JI(`igrafxrf.HLP',`CommandBarItems_Object')}

AddSeparator Method

Syntax *CommandBarItems*.**AddSeparator**([*Index* As Integer]) As CommandBarItem

Description The AddSeparator method adds a separator item onto a command bar. A separator is the line that separates items in a command bar or menu.

The *Index* argument is optional, and specifies the location at which to place the separator. If you omit the argument, the separator is placed at the end of the collection—the bottom of a menu, or the left side of a toolbar.

Example The following example creates a new menu called "Zoom". It then adds all the available zoom commands to the new menu. A separator line is added as the second item.

```
' Dimension the variables
Dim igxMenuBar As CommandBar
Dim igxPopup As CommandBarItem
' Get the MenuBar object
Set igxMenuBar = CommandBars.MenuBar
' Add a new menu to the MenuBar
Set igxPopup = igxMenuBar.CommandBarItems.AddPopup()
' Call the new menu "Zoom"
igxPopup.Caption = "Zoom"
' Add all the zoom tools to our new menu
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomComboBox)
igxPopup.Popup.CommandBarItems.AddSeparator
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomIn)
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomOut)
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomPrevious)
igxPopup.Popup.CommandBarItems.AddBuiltIn (ixZoomTool)
MsgBox "New Zoom menu is ready."
```

Find Method

Syntax *CommandBarItems*.**Find**(*Caption* As String) As CommandBarItem

Description The Find method searches the CommandBarItems collection to find a toolbar whose name (Caption property) matches the text string supplied in the *Caption* argument. If the command bar item is not found, the method returns a CommandBarItem object whose value is 'Nothing'. It is always important to check the returned object for the 'Nothing' value.

Example The following example displays a message box indicating whether or not the returned CommandBarItem object is valid.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
Dim igxCmdBarItem As CommandBarItem
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems object from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
' Get the Edit command bar item from the command bar
Set igxCmdBarItem = igxCmdBarItems.Find("Save")
' Display correct message box depending on object returned
If Not (igxCmdBarItem Is Nothing) Then
    MsgBox ("The command bar item was found.")
Else
    MsgBox ("The command bar item was not found.")
End If
```

See Also CommandBarItem object

{button CommandBarItems object,JI(`igrafxrf.HLP',`CommandBarItems_Object')}

FindBuiltinItem Method

Syntax *CommandBarItems*.**FindBuiltinItem**(*Item* As IxBuiltInCommand) As CommandBarItem

Description The FindBuiltinItem method searches the CommandBarItems collection to find iGrafx Professional built-in command bar items. The item to search for is specified by the *Item* argument, which must be one of the IxBuiltInCommand constants. If found, the method returns the CommandBarItem object for the specified built-in command. If the item is not found, the returned object contains the 'Nothing' value. It is always important to check the returned object for a value of 'Nothing'.

For the IxBuiltInCommand constant, refer to the Application.ExecuteCommand method.

Example The following example code displays a message box indicating whether or not the returned command bar item object is valid.

```
' Dimension the variables
Dim igxCmdBars As CommandBars
Dim igxCmdBar As CommandBar
Dim igxCmdBarItems As CommandBarItems
Dim igxCmdBarItem As CommandBarItem
' Get the CommandBars collection from the Application object
Set igxCmdBars = Application.CommandBars
' Get the first command bar from the CommandBars collection
Set igxCmdBar = igxCmdBars.Item(1)
' Get the CommandBarItems object from the CommandBar object
Set igxCmdBarItems = igxCmdBar.CommandBarItems
' Get the FileSave command bar item from the command bar
Set igxCmdBarItem = igxCmdBarItems.FindBuiltInItem(ixFileSave)
' Display correct message box depending on object returned
If Not (igxCmdBarItem Is Nothing) Then
    MsgBox ("The command bar item was found.")
Else
    MsgBox ("The command bar item was not found.")
End If
```

See Also CommandBarItem object

{button CommandBarItems object,JI(`igrafxrf.HLP',`CommandBarItems_Object')}

Item Method

Syntax *CommandBarItems*.**Item**(*Index* As Integer) As CommandBarItem

Description The Item method returns the CommandBarItem object at the specified *Index* from the CommandBarItems collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type CommandBarItem. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. Use error trapping if your code could potentially supply an invalid value for the index.

Example The following example uses the Item method to list the ID's and captions of all the CommandBarCommands on the File Menu.

```
' Dimension the variable
Dim strText As String
' Gather IDs and Captions for all items in the File Menu
With CommandBars.FindBuiltIn(ixFileMenu).CommandBarItems
   For Index = 1 To .Count
      If .Item(Index).Type = ixItemButton Then
         strText = strText & .Item(Index).Command.ID & " - " _
            & .Item(Index).Caption & Chr(13)
      End If
   Next Index
End With
' Display the result
MsgBox "File Menu Items:" & Chr(13) & _
    "ID        Caption" & Chr(13) & strText
```

See Also CommandBarItem object

{button CommandBarItems object,JI(`igrafxrf.HLP',`CommandBarItems_Object')}

CommandCategory Object

The CommandCategory object represents a categorized group of commands. iGrafx Professional has a number of built-in command categories, such as File, Edit, View, Insert, Tools, etc. The available command categories are listed in the Tools—Customize dialog. The individual items within each category are the commands, such as File—Print, Edit—Copy, etc., or are popups such as as File—New, which lead to additional commands.

The CommandCategory object is accessed through the Item method of the CommandCategories collection. The CommandCategories collection is accessed from the Application object. This means that all command categories are associated with the application, not with a document or a diagram.

From each CommandCategory object, you have access to a CommandBarItems collection, which is specifically associated only with that category. This collection stores the members of of the category. These members can be built-in command bar items, or ones you create.

Properties, Methods, and Events

All of the properties, methods, and events for the CommandCategory object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
BuiltIn		
CommandBarItems		
Name		
Parent		

Related Topics

CommandBarItem object
CommandBarItems object
CommandCategories object
iGrafx API Object Hierarchy

BuiltIn Property

Syntax *CommandCategory*.**BuiltIn**[= {**True | False**}]

Data Type Boolean (read-only)

Description The BuiltIn property specifies whether a category of commands is built in to the iGrafx Professional application, or whether the category is one that you or another developer has created.

Example The following example retrieves the CommandCategories collection for the application, and iterates through its objects to determine which ones are built-ins, and which ones are custom, and how many of each.

```
' Dimension the variables
Dim igxCmdCategories As CommandCategories
Dim igxCmdCategory As CommandCategory
Dim iBuiltIns As Integer
Dim iCustom As Integer
Dim sBuiltInList As String
Dim sCustomList As String
' Get the CommandCategories collection from the Application object
Set igxCmdCategories = Application.CommandCategories
iBuiltIns = 0
iCustom = 0
sBuiltInList = ""
sCustomList = ""
For iCount = 1 To igxCmdCategories.Count
    Set igxCmdCategory = igxCmdCategories.Item(iCount)
    If (igxCmdCategory.BuiltIn) Then
        iBuiltIns = iBuiltIns + 1
        sBuiltInList = sBuiltInList & igxCmdCategory.Name _
            & " is a built-in category." & Chr(13)
    Else
        iCustom = iCustom + 1
        sCustomList = sCustomList & igxCmdCategory.Name _
            & " is a custom category." & Chr(13)
    End If
Next iCount
If (iBuiltIns <> 0) Then
    MsgBox "There are " & iBuiltIns & " built-in command " _
        & "categories. They are:" & Chr(13) & sBuiltInList
Else
    MsgBox "No built-in command categories in the collection."
End If
If (iCustom <> 0) Then
    MsgBox "There are " & iCustom & " custom command " _
        & "categories. They are:" & Chr(13) & sCustomList
Else
    MsgBox "No custom command categories in the collection."
End If
```

{button CommandCategory object,JI(`igrafxrf.HLP',`CommandCategory_Object')}

CommandBarItems Property

Syntax *CommandCategory*.**CommandBarItems**

Data Type CommandBarItems collection object (read-only, See Object Properties)

Description The CommandBarItems property returns the CommandBarItems collection for the specified CommandCategory object. The CommandBarItems collection contains all of the CommandBarItem objects that have been defined for the command category. Through the CommandBarItem(s) objects, you can add and delete items, or make modifications to the items in the category.

Example The following example retrieves the CommandCategories collection and gets the first command category. It then gets the first category's CommandBarItems collection, and lists all the items that are members of the first command category.

```
' Dimension the variables
Dim igxCmdCategories As CommandCategories
Dim igxCmdCategory As CommandCategory
Dim igxCmdBarItem As CommandBarItem
Dim iCount As Integer
Dim sList As String
' Get the CommandCategories collection from the Application object
Set igxCmdCategories = Application.CommandCategories
' Get the first command category
Set igxCmdCategory = igxCmdCategories.Item(1)
' List all the command bar items that are members of the first
' command category
sList = ""
For iCount = 1 To igxCmdCategory.CommandBarItems.Count
    Set igxCmdBarItem = igxCmdCategory.CommandBarItems.Item(iCount)
    sList = sList & igxCmdBarItem.Caption & Chr(13)
Next iCount
' Display the list
MsgBox "The first command category in the collection is: " _
    & igxCmdCategory.Name & Chr(13) & "The items it contains are:" _
    & Chr(13) & sList
```

See Also CommandBarItem object

CommandBarItems object

iGrafx API Object Hierarchy

{button CommandCategory object,JI(`igrafxrf.HLP',`CommandCategory_Object')}

CommandCategories Object

The CommandCategories object is a collection of individual CommandCategory objects. There is one CommandCategories collection, whcih is associated with the Application object. Its purpose is to store and provide access to the individual CommandCategory objects that have been defined for the application.

The individual command categories can be viewed through the interface by selecting Customize from the Tools menu. The categories list in this dialog is the CommandCategories collection.

The CommandCategories object provides the following functionality:

- The ability to access the CommandCategory objects in the collection.
- The ability to determine how many CommandCategory objects are currently in the collection.
- The ability to find a CommandCategory object in the collection based on the CommandCategory.Name property.
- The ability to add a new CommandCategory object to the collection.

Each iGrafx Professional built-in command category (File, Edit, View, Tools, etc.) is stored in the collection. Additionally, you can add new, custom categories that can consist of any combination of built-in command bar items or custom command bar items created by you or some other developer.

Properties, Methods, and Events

All of the properties, methods, and events for the CommandCategories object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Find	
Parent	Item	

Related Topics

CommandCategory object

iGrafx API Object Hierarchy

Add Method

Syntax *CommandCategories*.**Add** (*CategoryName* As String) As CommandCategory

Description The Add method adds a new command category to the CommandCategories collection. You specify a name for the new command category with the *CategoryName* argument. The method returns the new CommandCategory object.

Example The following example shows how to add a new command category called ‘MyCategory’ to the CommandCategories collection.

```
' Dimension the variables
Dim igxCmdCategories As CommandCategories
Dim igxCmdCategory As CommandCategory
' Get CommandCategories object from the Application object
Set igxCmdCategories = Application.CommandCategories
' Add the command category to command categories collection
Set igxCmdCategory = igxCmdCategories.Add("MyCategory")
```

See Also CommandCategory object

{button CommandCategories object,JI(`igrafxrf.HLP',`CommandCategories_Object')}

Find Method

Syntax *CommandCategories*.**Find** (*CategoryName* As String) As CommandCategory

Description The Find method searches the CommandCategories collection for a command category whose name matches that supplied by the *CategoryName* argument. If a match is found, the method returns the specified CommandCategory object. If no match is found, then the methods returns a CommandCategory object whose value is 'Nothing'.

Example The following example uses the Find method to search the CommandCategories collection for a command category named "File". A message box is displayed indicating success or failure.

```
' Dimension the variables
Dim igxCmdCategories As CommandCategories
Dim igxCmdCategory As CommandCategory
' Get CommandCategories object from the Application object
Set igxCmdCategories = Application.CommandCategories
' Find the "File" command category
Set igxCmdCategory = igxCmdCategories.Find("File")
' Test returned object and display appropriate message box
If igxCmdCategory Is Nothing Then
    MsgBox "Command Category not found"
Else
    MsgBox igxCmdCategory.Name & " command category was found."
End If
```

See Also CommandCategory object

{button CommandCategories object,JI(`igrafxrf.HLP',`CommandCategories_Object')}

Item Method

Syntax *CommandCategories*.**Item**(*Index* As Integer) As CommandCategory

Description The Item method returns the CommandCategory object at the specified *Index* from the CommandCategories collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type CommandCategory. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example retrieves the CommandCategories collection for the application, and iterates through its objects (using the Item method) to determine which ones are built-ins, and which ones are custom, and how many of each.

```
' Dimension the variables
Dim igxCmdCategories As CommandCategories
Dim igxCmdCategory As CommandCategory
Dim iBuiltIns As Integer
Dim iCustom As Integer
Dim sBuiltInList As String
Dim sCustomList As String
' Get the CommandCategories collection from the Application object
Set igxCmdCategories = Application.CommandCategories
iBuiltIns = 0
iCustom = 0
sBuiltInList = ""
sCustomList = ""
For iCount = 1 To igxCmdCategories.Count
    Set igxCmdCategory = igxCmdCategories.Item(iCount)
    If (igxCmdCategory.BuiltIn) Then
        iBuiltIns = iBuiltIns + 1
        sBuiltInList = sBuiltInList & igxCmdCategory.Name _
            & " is a built-in category." & Chr(13)
    Else
        iCustom = iCustom + 1
        sCustomList = sCustomList & igxCmdCategory.Name _
            & " is a custom category." & Chr(13)
    End If
Next iCount
If (iBuiltIns <> 0) Then
    MsgBox "There are " & iBuiltIns & " built-in command " _
        & "categories. They are:" & Chr(13) & sBuiltInList
Else
    MsgBox "No built-in command categories in the collection."
End If
If (iCustom <> 0) Then
    MsgBox "There are " & iCustom & " custom command " _
        & "categories. They are:" & Chr(13) & sCustomList
Else
    MsgBox "No custom command categories in the collection."
End If
```

See Also CommandCategory object

{button CommandCategories object,JI(`igrafxrf.HLP',`CommandCategories_Object')}

CommandHandler Object

The CommandHandler object is the interface that is written by a developer to implement custom CommandBarItem objects.

The CommandHandler object is used to create a user-defined command structure. A command handler is created in a user-definable class. Use the following steps to create your own command handler.

1 Create a new class module.
2 Change the name of the class from Class1 to MyCommandClass.
3 Type the following code into the class module:

```
Implements CommandHandler

Private Sub CommandHandler_Execute()
' This section is fired when the command is executed.
End Sub

Private Sub CommandHandler_Help()
' This section is fired when the command is active and the
' help system is invoked.
End Sub

Private Sub CommandHandler_Update(ByVal Command As IXCommandBarCommand)
' This is the section that handles when an update is sent to the
' command. This is useful when you want to use the update to enable
' and disable commands, or if they are in a menu, to check and uncheck
' items in the menu. It is necessary to enable the command unless you
' are going to enable it in the future through some other means.
' Commands are normally disabled unless enabled.
Command.Enabled = True
End Sub
```

If an API needs a command handler, just use the New MyCommandClass statement. This statement creates a new “MyCommandClass” object, and nothing else needs to be done. An example that adds the command to the CommandBarItems collection is shown below.

```
CommandBar.Items.AddCommand "Edit Code...", New MyCommandClass, 1
```

Properties, Methods, and Events

All of the properties, methods, and events for the CommandHandler object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
	Execute	
	Help	
	Update	

Execute Method

Syntax *CommandHandler*.**Execute**

Description The Execute method is called when the user clicks on a custom CommandBarItem; or in other words, when the associated button or menu item is selected or the CommandBarCommand.Execute method is used. This method calls your CommandHandler_Execute subroutine.

{button CommandHandler object,JI(`igrafxrf.HLP',`CommandHandler_Object')}

Help Method

Syntax *CommandHandler*.**Help**

Description The Help method is called when Context Sensitive help is needed for your custom CommandBarItem. This method calls your CommandHandler_Help subroutine.

{button CommandHandler object,JI(`igrafxrf.HLP',`CommandHandler_Object')}

Update Method

Syntax *CommandHandler*.**Update** *Command* As CommandBarCommand

Description The Update method is called when the application updates its user interface. This method calls your CommandHandler_Update subroutine.

{button CommandHandler object,JI(`igrafxrf.HLP',`CommandHandler_Object')}

Cursor Object

The Cursor object controls the display of the cursor for the application. The Cursor object is subordinate to, and accessed from, the Application object. With this object, the developer can control the type of cursor to display, and its position relative to the Windows desktop. Typically, you would use the Cursor object to show an Hourglass cursor during a long operation.

Properties, Methods, and Events

All of the properties, methods, and events for the Cursor object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Parent		
Type		
XPosition		
YPosition		

Related Topics

Application.Cursor property

iGrafx API Object Hierarchy

Type Property

Topic Under Construction!!!

Syntax *Cursor*.**Type**

Data Type IxCursorType enumerated constant (read/write)

Description The Type property specifies the type of cursor to display in the application interface.

The IxCursorType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixCursorNormal
1	ixCursorHourglass
2	ixCursorIBeam

Example The following example changes the cursor to a wait cursor (hourglass) while processing, then restores it to a normal cursor after the processing is complete.

Example Under Construction!!! Do NOT Use

```
' Dimension the variables
Dim igxCursor As Cursor
Dim igxPGauge As PercentGauge
Dim iCount As Long
' Get the Cursor and percent gauge objects
Set igxCursor = Application.Cursor
Set igxPGauge = Application.PercentGauge
' Set PercentGauge properties
igxPGauge.CancelEnabled = True
igxPGauge.Caption = "PercentGauge Test"
igxPGauge.Text = "This is the top text."
igxPGauge.Text2 = "This is the bottom text."
igxPGauge.Visible = True
' Change the cursor to an hourglass
igxCursor.Type = ixCursorHourglass
' Create values and set them into the value property
' of the percent gauge
MsgBox "Click OK to start the Percent Gauge test"
For iCount = 1 To 100000
    igxPGauge.Value = Int(iCount / 1000)
    Application.RefreshUI
Next iCount
' Hide the percent gauge when done
igxPGauge.Visible = False


' Dimension the variables
Dim igxCursor As Cursor
Dim igxPGauge As PercentGauge
Dim iCount As Long
```

```
' Get the Cursor and percent gauge objects
Set igxCursor = Application.Cursor
Set igxPGauge = Application.PercentGauge
' Set PercentGauge properties
igxPGauge.CancelEnabled = True
igxPGauge.Caption = "PercentGauge Test"
igxPGauge.Visible = True
' Create values and set them into the value property
' of the percent gauge
MsgBox "Click OK to start the Percent Gauge test"
igxCursor.Type = ixCursorHourglass
DoEvents
For iCount = 1 To 100000
    igxPGauge.Value = Int(iCount / 1000)
    Application.RefreshUI
Next iCount
' Hide the percent gauge when done
igxPGauge.Visible = False
igxCursor.Type = ixCursorNormal
```

{button Cursor object,JI(`igrafxrf.HLP',`Cursor_Object')}

XPosition Property

Syntax *Cursor*.**XPosition**

Data Type Long (read-only)

Description The XPosition property returns the position of the cursor in the X direction. The units for this property are pixels, and are relative to the Windows desktop. This property returns the same value as the Windows API call GetCursorPOS.

Example The following example outputs the XPosition and YPosition of the cursor to the Output window every time the code is executed.

```
' Dimension the variables
Dim igxCursor As Cursor
' Get the Cursor object from the Application object
Set igxCursor = Application.Cursor
'  Output the cursor position to the Output window
Output " Cursor XPosition =" & Str(igxCursor.XPosition) _
    & ", YPosition =" & Str(igxCursor.YPosition)
```

{button Cursor object,JI(`igrafxrf.HLP',`Cursor_Object')}

YPosition Property

Syntax *Cursor*.**YPosition**

Data Type Long (read-only)

Description The YPosition property returns the position of the cursor in the Y direction. The units for this property are pixels, and are relative to the Windows desktop. This property returns the same value as the Windows API call GetCursorPOS.

Example The following example outputs the XPosition and YPosition of the cursor to the Output window every time the code is executed.

```
' Dimension the variables
Dim igxCursor As Cursor
' Get the Cursor object from the Application object
Set igxCursor = Application.Cursor
' Output the cursor position to the Output window
Output "Cursor XPosition =" & Str(igxCursor.XPosition) _
    & ", YPosition =" & Str(igxCursor.YPosition)
```

{button Cursor object,JI(`igrafxrf.HLP',`Cursor_Object')}

PopupWindow Object

The PopupWindow object represents either a floating or a docking window. The PopupWindow object can represent any of the following, depending on the value of the Type property:

- The Gallery window
- The Components window
- The Properties window
- The VBA Properties window
- The Output window
- The Note window
- The Entity Manager window

Of these various window types, the Gallery window and the Output window can dock to the main application window. The rest do not.

The PopupWindow object gives the programmer the ability to get and set various properties to manipulate the PopupWindow object's placement and visibility on the screen relative to the iGrafx Professional application.

Properties, Methods, and Events

All of the properties, methods, and events for the PopupWindow object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Close	
Collapsed		
Dockable		
Height		
Left		
Parent		
Position		
Top		
Type		
Visible		
Width		

Related Topics

PopupWindows object

iGrafx API Object Hierarchy

Close Method

Syntax *PopupWindow*.**Close**

Description The Close method closes the specified PopupWindow object, removing it from the interface.

Example The following example opens the Components dialog and then modifies it's position and size. It then closes the window.

```
' Dimension the variables
Dim igxPopup As PopupWindow
' Open the Components dialog box
MsgBox "Click OK to open the Components dialog box"
CommandBars.FindBuiltIn(ixFileMenu).CommandBarItems _
  .FindBuiltInItem(ixFileComponents).Command.Execute
' Work with current popup windows
With Application.PopupWindows
    For Index = 1 To .Count
        If .Item(Index).Type = ixPopupComponents Then
            Set igxPopup = .Item(Index)
        End If
    Next Index
End With
' Move the window to the top of the screen
MsgBox "Click OK to move the window top screen"
igxPopup.Top = 10
igxPopup.Left = 100
' Stretch the height of the window
MsgBox "Click OK to stretch the height"
igxPopup.Height = igxPopup.Height * 1.5
MsgBox "Click OK to close the window"
igxPopup.Close
MsgBox "Click OK to continue"
```

{button PopupWindow object,JI(`igrafxrf.HLP',`PopupWindow_Object')}

Collapsed Property

Syntax *PopupWindow*.**Collapsed**[= {**True | False**}]

Data Type Boolean (read-only)

Description The Collapsed property indicates whether the specified popup window is collapsed, meaning only its title bar is shown. Only dockable windows can be collapsed. Currently, only Gallery and Output windows are dockable (Type property equals ixPopupGallery or ixPopupOutput, respectively).

Example The following example reports whether the Gallery window is collapsed or not.

```
' Dimension the variables
Dim igxPopup As PopupWindow
' Open the Gallery window
Application.Gallery.Visible = True
' Find the gallery popup window
With Application.PopupWindows
   For Index = 1 To .Count
      If .Item(1).Type = ixPopupGallery Then
         Set igxPopup = .Item(1)
      End If
   Next Index
End With
If igxPopup.Collapsed = True Then
   MsgBox "The Gallery window is collapsed."
Else
   MsgBox "The Gallery windows is not collapsed."
End If
```

{button PopupWindow object,JI(`igrafxrf.HLP',`PopupWindow_Object')}

Dockable Property

Syntax *PopupWindow*.**Dockable**[= {**True | False**}]

Data Type Boolean (read-only)

Description The Dockable property indicates whether the specified popup window is able to dock to the application window. Currently, only PopupWindows of type ixPopupGallery or ixPopupOuput are dockable.

Example The following example reports whether a popup window is dockable.

```
' Dimension the variables
Dim igxPopup As PopupWindow
' Open the Gallery window
Application.Gallery.Visible = True
' Find the gallery popup window
With Application.PopupWindows
   For Index = 1 To .Count
      If .Item(1).Type = ixPopupGallery Then
         Set igxPopup = .Item(1)
      End If
   Next Index
End With
If igxPopup.Dockable = True Then
   MsgBox "The window is dockable."
Else
   MsgBox "The window is not collapsed."
End If
```

{button PopupWindow object,JI(`igrafxrf.HLP',`PopupWindow_Object')}

Left Property

Syntax *PopupWindow*.**Left**

Data Type Integer (read/write)

Description The Left property specifies the position of the left edge of the PopupWindow object. The position is in pixels based on the display device—the computer screen. The position is in relation to the application window, where 0 (zero) represents the left-most edge of the application window. Positive values place the PopupWindow object farther to the right. Negative values place the PopupWindow off the left edge of the screen.

Large positive or negative values may place the window partially or completely off the edge of the screen. Keep your display device's dimensions in mind when using this property.

Example The following example opens the Components window, and then positions it using the Left and Top properties.

```
' Dimension the variables
Dim igxPopup As PopupWindow
' Open the Components dialog box
MsgBox "Click OK to open the Components dialog box"
CommandBars.FindBuiltIn(ixFileMenu).CommandBarItems _
  .FindBuiltInItem(ixFileComponents).Command.Execute
' Work with current popup windows
With Application.PopupWindows
   For Index = 1 To .Count
      If .Item(Index).Type = ixPopupComponents Then
         Set igxPopup = .Item(Index)
      End If
   Next Index
End With
' Move the window to the top of the screen
MsgBox "Click OK to move the window top screen"
igxPopup.Top = 10
igxPopup.Left = 100
' Stretch the height of the window
MsgBox "Click OK to stretch the height"
igxPopup.Height = igxPopup.Height * 1.5
MsgBox "Click OK to close the window"
igxPopup.Close
MsgBox "Click OK to continue"
```

{button PopupWindow object,JI(`igrafxrf.HLP',`PopupWindow_Object')}

Position Property

Syntax *PopupWindow*.**Position**

Data Type IxPosition enumerated constant (read/write)

Description The Position property specifies where to dock a dockable PopupWindow. Also, you can choose not to dock the window.

Of the current set of Popup window types, only the Gallery window and Output window are dockable, and therefore, affected by this property. Other Popup window types are not affected.

Changing the Position property's value moves the specified PopupWindow object. If set to ixFloating from a fixed position setting, the window becomes floating—undocked.

The IxPosition constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixDockLeft
1	ixDockTop
2	ixDockRight
3	ixDockBottom
4	ixFloating

Example The following example docks the Gallery window in each of the four positions.

```
' Dimension the variables
Dim igxPopup As PopupWindow
' Open the Gallery window
Application.Gallery.Visible = True
' Find the gallery popup window
With Application.PopupWindows
   For Index = 1 To .Count
      If .Item(1).Type = ixPopupGallery Then
         Set igxPopup = .Item(1)
      End If
   Next Index
End With
MsgBox "Dock the gallery left"
igxPopup.Position = ixDockLeft
MsgBox "Dock the gallery right"
igxPopup.Position = ixDockRight
MsgBox "Dock the gallery top"
igxPopup.Position = ixDockTop
MsgBox "Dock the gallery bottom"
igxPopup.Position = ixDockBottom
MsgBox "Click OK to continue"
igxPopup.Position = ixDockRight
```

{button PopupWindow object,JI(`igrafxrf.HLP',`PopupWindow_Object')}

Type Property

Syntax *PopupWindow*.**Type**

Data Type IxPopupWindowType enumerated constant (read/write)

Description The Type property returns the type of the specified PopupWindow object.

The types of PopupWindow objects are the Gallery window, the Components window, the Properties window, the VBA Properties window, the Output window, the Note window and the Entity Manager window. Each type of window has an associated IxPopupWindowType that is listed in the table below.

The IxPopupWindowType constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant
0	ixPopupGallery
1	ixPopupComponents
2	ixPopupProperties
3	ixPopupVBAProperties
4	ixPopupOutput
5	ixPopupNote
6	ixPopupEntityManager

Example The following example finds the Gallery window using the Type property. It then docks the gallery in each of the four docking positions.

```
' Dimension the variables
Dim igxPopup As PopupWindow
' Open the Gallery window
Application.Gallery.Visible = True
' Find the gallery popup window
With Application.PopupWindows
   For Index = 1 To .Count
      If .Item(1).Type = ixPopupGallery Then
         Set igxPopup = .Item(1)
      End If
   Next Index
End With
MsgBox "Dock the gallery left"
igxPopup.Position = ixDockLeft
MsgBox "Dock the gallery right"
igxPopup.Position = ixDockRight
MsgBox "Dock the gallery top"
igxPopup.Position = ixDockTop
MsgBox "Dock the gallery bottom"
igxPopup.Position = ixDockBottom
MsgBox "Click OK to continue"
igxPopup.Position = ixDockRight
```

{button PopupWindow object,JI(`igrafxrf.HLP',`PopupWindow_Object')}

PopupWindows Object

The PopupWindows object is a collection of the currently open PopupWindow objects (floating and/or docking windows). A PopupWindows collection is associated with and accessible from the Application object.

The PopupWindows object provides the following functionality:

- The ability to access any PopupWindow objects that are currently open within the application.
- The ability to determine how many PopupWindow objects are currently in the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the PopupWindows object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
Parent		

Related Topics

PopupWindow object

iGrafx API Object Hierarchy

Item Method

Syntax *PopupWindows*.**Item**(*Index* As Integer) As PopupWindow

Description The Item method returns the PopupWindow object at the specified *Index* from the PopupWindows collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type PopupWindow. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example gets the applications PopupWindows collection, and iterates through it, reporting the type of each popup window it contains.

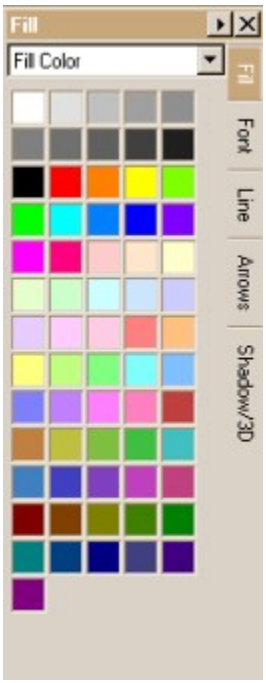
```
' Dimension the variables
Dim iCount As Integer
Dim igxPopup As PopupWindows
Set igxPopup = Application.PopupWindows
' Iterate through collection and print name
' of each PopupWindow object into the
' iGrafx Output Window using the Item method
For idx = 1 To igxPopup.Count
   Select Case igxPopup.Item(idx).Type
       Case ixPopupGallery
           Output "Gallery PopupWindow"
       Case ixPopupComponents
           Output "Components PopupWindow"
       Case ixPopupProperties
           Output "Properties PopupWindow"
       Case ixPopupVBAProperties
           Output "VBAProperties PopupWindow"
       Case ixPopupOutput
           Output "Output PopupWindow"
       Case ixPopupNote
           Output "Note PopupWindow"
       Case ixPopupEntityManager
           Output "EntityManager PopupWindow"
   End Select
Next iCount
Output "The above are currently in the PopupWindows collection."
```

{button PopupWindows object,JI(`igrafxrf.HLP',`PopupWindows_Object')}

Gallery Object

The Gallery object represents the Gallery docking window. This object provides control for minimizing the Gallery window, and for accessing the GalleryPanes collection, which in turn allows access to individual GalleryPane objects. The Gallery object is accessed from the Application object, and there is only one Gallery object for the application.

The following illustration shows what the Gallery looks like.



Properties, Methods, and Events

All of the properties, methods, and events for the Gallery object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Collapsed		
GalleryPanes		
Parent		
Visible		

Related Topics

GalleryPane object
GalleryPanes object
PopupWindow object
iGrafx API Object Hierarchy

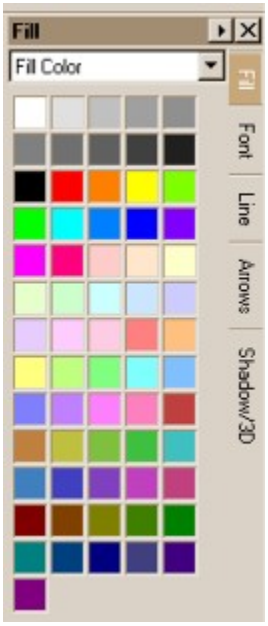
Collapsed Property

Syntax *Gallery*.**Collapsed**[= {**True** | **False**}]

Data Type Boolean (read/write)

Description The Collapsed property collapses the Gallery window. It allows you to programmatically alter the appearance of the Gallery object such that it is shown in either a normal or collapsed state.

The following illustration shows the gallery in its normal and collapsed state.



Example The following example toggles the Collapsed property for the gallery object.

```
' Dimension the variables
Dim igxGallery As Gallery
' Get the Gallery object
Set igxGallery = Application.Gallery
MsgBox "Click OK to collapse the Gallery window"
```

```
' Set Collapsed property to True
igxGallery.Collapsed = True
MsgBox "Click OK to show the Gallery window"
igxGallery.Collapsed = False
MsgBox "View the diagram"
```

{button Gallery object,JI(`igrafxrf.HLP',`Gallery_Object')}

GalleryPanes Property

Syntax *Gallery*.**GalleryPanes**

Data Type GalleryPanes collection object (read-only, See Object Properties)

Description The GalleryPanes property returns the GalleryPanes collection for the application's Gallery object. Each GalleryPane can contain various tools and options for constructing and formatting a diagram.

Example The following example accesses the GalleryPanes collection of the Gallery object, and displays the number of sub-panes for each GalleryPane object.

```
' Dimension the variables
Dim igxGallery As Gallery
Dim igxGalleryPanes As GalleryPanes
' Get the Gallery and GalleryPanes objects
Set igxGallery = Application.Gallery
Set igxGalleryPanes = igxGallery.GalleryPanes
For iCount = 1 To igxGalleryPanes.Count
    MsgBox "Gallery Pane " & iCount & " has " _
        & igxGalleryPanes.Item(iCount).SubPaneCount _
        & " sub-panes."
Next iCount
```

See Also GalleryPane object

GalleryPanes object

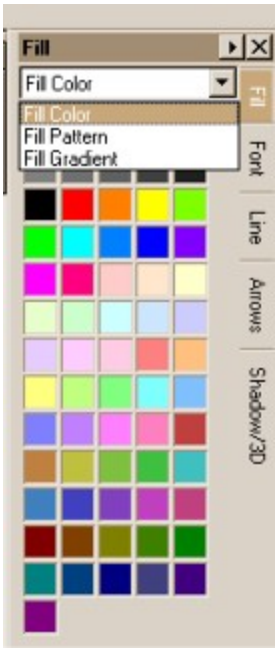
iGrafx API Object Hierarchy

{button Gallery object,JI(`igrafxrf.HLP',`Gallery_Object')}

GalleryPane Object

The GalleryPane object represents a tab of the Gallery window. Each GalleryPane can contain various tools and options for constructing and formatting a diagram. A GalleryPane also can have subpanes. The GalleryPane object allows you to ascertain the number of sub-panes that a GalleryPane object has access to, and provides the ability to make a certain sub-pane the active sub-pane.

The following illustration shows that the Gallery window has five gallery panes: Fill, Font, Line, Arrows, and Shadow/3D. Also, as shown by the combo box, the Fill GalleryPane object has three subpanes: Fill Color, Fill Pattern, and Fill Gradient.



Properties, Methods, and Events

All of the properties, methods, and events for the GalleryPane object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Active	Activate	
Application		
Parent		
SubPaneCount		
SubPaneIndex		
Visible		

Related Topics

GalleryPanes object

iGrafx API Object Hierarchy

Activate Method

Syntax *GalleryPane*.**Activate**

Description The Activate method activates the specified GalleryPane object.

Example The following example activates each gallery pane object in the galleryPanes collection.

```
' Dimension the variables
Dim igxGallery As Gallery
Dim igxGalleryPanes As GalleryPanes
' Get the Gallery and GalleryPanes objects
Set igxGallery = Application.Gallery
Set igxGalleryPanes = igxGallery.GalleryPanes
For iCount = 1 To igxGalleryPanes.Count
    Call igxGalleryPanes.Item(iCount).Activate
    MsgBox "Gallery Pane " & iCount & " has been activated."
Next iCount
```

See Also Active property

{button GalleryPane object,JI(`igrafxrf.HLP',`GalleryPane_Object')}

Active Property

Syntax *GalleryPane*.**Active**[= {**True** | **False**}]

Data Type Boolean (read-only)

Description The Active property indicates whether the specified GalleryPane object is the active pane.

In the following illustration, the "Fill" GalleryPane is the active pane as indicated by the highlighted tab and the caption of the Gallery.

Example The following example uses the GalleryPane.Active property to determine which gallery pane is currently active, and then activates the next gallery pane based on the user's response until all panes have been activated once.

```
' Dimension the variables
Dim igxGallery As Gallery
Dim igxGalleryPanes As GalleryPanes
' Get the Gallery and GalleryPanes objects
Set igxGallery = Application.Gallery
Set igxGalleryPanes = igxGallery.GalleryPanes
' Find the currently active gallery pane, and step through
' activating the rest until return to the original
For iCount = 1 To igxGalleryPanes.Count
    If (igxGalleryPanes.Item(iCount).Active) Then
        iStart = iCount
        For Idx = 1 To igxGalleryPanes.Count - iStart
            Call igxGalleryPanes.Item(Idx + iStart).Activate
            MsgBox "Gallery Pane " & Idx + iStart & " has been " _
                & "activated." & Chr(13) _
                & "Activate the next gallery pane."
        Next Idx
        For Idx = 1 To iStart
            Call igxGalleryPanes.Item(Idx).Activate
            MsgBox "Gallery Pane " & Idx & " has been activated." _
                & Chr(13) & "Activate the next gallery pane."
```

```
            Next Idx
            Exit For
        End If
    Next iCount
```

See Also Activate method

{button GalleryPane object,JI(`igrafxrf.HLP',`GalleryPane_Object')}

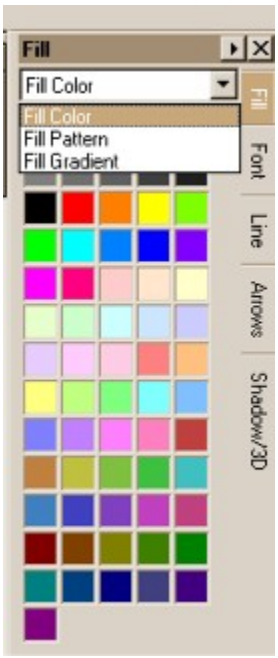
SubPaneCount Property

Syntax *GalleryPane*.**SubPaneCount**

Data Type Integer (read-only)

Description The SubPaneCount property returns the number of sub-panes that a GalleryPane object contains.

The following illustration shows the "Fill" GalleryPane, which has three sub-panes: Fill Color, Fill Pattern, and Fill Gradient.



Example The following example accesses the GalleryPanes collection of the Gallery object, and displays the number of sub-panes for each GalleryPane object.

```
' Dimension the variables
Dim igxGallery As Gallery
Dim igxGalleryPanes As GalleryPanes
' Get the Gallery and GalleryPanes objects
Set igxGallery = Application.Gallery
Set igxGalleryPanes = igxGallery.GalleryPanes
For iCount = 1 To igxGalleryPanes.Count
    MsgBox "Gallery Pane " & iCount & " has " _
        & igxGalleryPanes.Item(iCount).SubPaneCount _
        & " sub-panes."
Next iCount
```

See Also SubPaneIndex property

{button GalleryPane object,JI(`igrafxrf.HLP','GalleryPane_Object')}

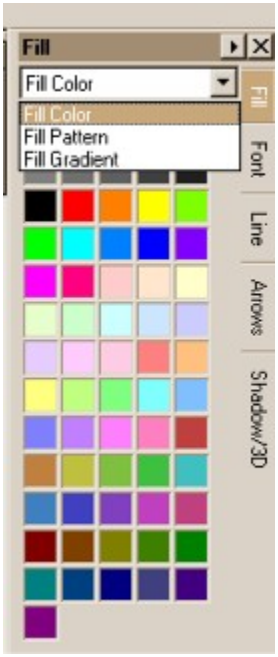
SubPaneIndex Property

Syntax *GalleryPane*.**SubPaneIndex**

Data Type Integer (read/write)

Description The SubPaneIndex property specifies which sub-pane is the active sub-pane.

The following illustration shows the “Fill” GalleryPane, which has three sub-panes: Fill Color, Fill Pattern, and Fill Gradient. The Fill Color sub-pane is the currently active sub-pane. Therefore, the SubPaneIndex would be 1, since Fill Color is the first listed sub-pane.



Example The following example accesses the GalleryPanes collection of the Gallery object. It then iterates through all of the gallery panes. If a gallery pane has sub-panes, the gallery pane is activated and each sub-pane is displayed.

```
' Dimension the variables
Dim igxGallery As Gallery
Dim igxGalleryPanes As GalleryPanes
' Get the Gallery and GalleryPanes objects
Set igxGallery = Application.Gallery
Set igxGalleryPanes = igxGallery.GalleryPanes
For iCount = 1 To igxGalleryPanes.Count
    If (igxGalleryPanes.Item(iCount).SubPaneCount > 0) Then
        igxGalleryPanes.Item(iCount).Activate
        For Idx = 1 To igxGalleryPanes.Item(iCount).SubPaneCount
            igxGalleryPanes.Item(iCount).SubPaneIndex = Idx
            MsgBox "Sub-pane " & Idx & " displayed."
        Next Idx
    End If
Next iCount
```

See Also SubPaneCount property

{button GalleryPane object,JI(`igrafxrf.HLP',`GalleryPane_Object')}

GalleryPanes Object

The GalleryPanes object is a collection of GalleryPane objects. A GalleryPanes collection is only associated with and accessible through the Gallery object. Its purpose is to provide access to the individual GalleryPane objects, and any sub-panes that may be defined for a gallery pane.

In the following illustration, the Gallery window has five GalleryPane objects. Note that the Gallery window may contain more than five GalleryPanes—for example, a new gallery pane is added each time a Share Media subject is opened.

The GalleryPanes object provides the following functionality:

- The ability to access any GalleryPane objects in the collection.
- The ability to determine how many GalleryPane objects are currently in the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the GalleryPanes object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
Parent		

Related Topics

Gallery object

GalleryPane object

iGrafx API Object Hierarchy

Item Method

Syntax *GalleryPanes*.**Item**(*Index* As Integer) As GalleryPane

Description The Item method returns the GalleryPane object at the specified *Index* from the GalleryPanes collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type GalleryPane. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example gets the GalleryPanes collection, and then prints to the Output window the number of sub-panes contained in each gallery pane, and the total number of sub-panes for all gallery panes in the collection.

```
' Dimension the variables
Dim iCount, iTotalSubPanes As Integer
Dim igxPanes As GalleryPanes
iTotalSubPanes = 0
Set igxPanes = Application.Gallery.GalleryPanes
' Iterate through GalleryPanes collection and
' print SubPaneCount for each GalleryPane object in
' the GalleryPanes collection
For iCount = 1 To igxPanes.Count
   iTotalSubPanes = iTotalSubPanes + _
       igxPanes.Item(iCount).SubPaneCount
   Output igxPanes.Item(iCount).SubPaneCount
Next iCount
' Output the total SubPaneCount of all GalleryPane objects
Output "Total sub-panes for all GalleryPane objects: " _
   & iTotalSubPanes
MsgBox "View the results."
```

{button GalleryPanes object,JI(`igrafxrf.HLP',`GalleryPanes_Object')}

Font Object

The Font object represents a font that is available for use within iGrafx Professional. With the Font object, a developer can read or set various attributes of a font, such as color, size, bold, italic, etc.

The following objects have a Font property:

- Field
- Legend
- OffPageConnectorFormat
- Diagram.IndicatorFont
- OffPageConnectorFormat.ToPageFont
- TextRange

Of these objects, TextRange deserves special mention. A TextRange can be any range of consecutive characters within a block of text. Since a range of characters may be formatted using different fonts, some properties of the Font object sometimes return a special value (ixUndefined) if multiple fonts are used in the text range.

Properties, Methods, and Events

All of the properties, methods, and events for the Font object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Bold		
Color		
Italic		
Name		
Opaque		
Parent		
Size		
Strikethrough		
Underline		

Related Topics

FontNames object

Bold Property

Syntax *Font*.**Bold**

Data Type Long (read/write)

Description The Bold property specifies whether to set the weight of the Font object to bold. A value of zero equals False; any non-zero value equals True.

In the case of a TextRange, which can use more than one font, this property can return the value of ixUndefined (99999999) if conflicting font specifications are made for the same characters. If a TextRange object only uses one font, then the return value is True (non-zero) or False (0).

Example The following example creates a shape in the active diagram, and adds some text to the shape. The shape's TextRange object is retrieved through the TextBlock object, and the entire text string is assigned. Then the font weight of the shape's text is changed to bold.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxTextRange As TextRange
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape1.Text = "Some shape text"
MsgBox "View the shape before changing the font characteristics"
' Get the shape's TextRange object
Set igxTextRange = igxShape1.TextBlock.TextRange
igxTextRange.Font.Bold = 1
MsgBox "View the results"
```

{button Font object,JI(`igrafxrf.HLP',`Font_Object')}

Italic Property

Syntax *Font*.**Italic**

Data Type Long (read/write)

Description The Italic property specifies whether to set the angle of the Font object to italic. A value of zero equals False; any non-zero value equals True.

In the case of a TextRange, which can use more than one font, this property can return the value of ixUndefined (99999999) if conflicting font specifications are made for the same characters. If a TextRange object only uses one font, then the return value is True (non-zero) or False (0).

Example The following example creates a shape in the active diagram, and adds some text to the shape. The shape's TextRange object is retrieved through the TextBlock object, and the entire text string is assigned. Then the font angle of the shape's text is changed to italic.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxTextRange As TextRange
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape1.Text = "Some shape text"
MsgBox "View the shape before changing the font characteristics"
' Get the shape's TextRange object
Set igxTextRange = igxShape1.TextBlock.TextRange
igxTextRange.Font.Italic = 1
MsgBox "View the results"
```

{button Font object,JI(`igrafxrf.HLP',`Font_Object')}

Opaque Property

Syntax *Font*.**Opaque**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Opaque property specifies whether the font is displayed as opaque against the background. If the property is set to False, the text is set against the foreground color of the iGrafx Professional object. If the property is set to True, then the text is 'blocked' against the Windows background color.

Example The following example creates a shape in the active diagram, and adds some text to the shape. The shape's TextRange object is retrieved through the TextBlock object, and the entire text string is assigned. Then the Opaque property is set to True. This change does not produce a visible effect until the shape's fill color is changed.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxTextRange As TextRange
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape1.Text = "Some shape text"
MsgBox "View the shape before changing the font characteristics"
' Get the shape's TextRange object
Set igxTextRange = igxShape1.TextBlock.TextRange
igxTextRange.Font.Opaque = True
MsgBox "Opaque text set to True"
' Change the fill of the shape to red
igxShape1.FillColor = vbRed
MsgBox "Opaque text now shows because shape's fill is not white"
```

{button Font object,JI(`igrafxrf.HLP',`Font_Object')}

Size Property

Syntax *Font*.**Size**

Data Type Double (read/write)

Description The Size property specifies the size of the Font object, in units of ‘‘points’’.

In the case of a TextRange, which can use more than one font, this property can return the value of ixUndefined (99999999) if conflicting font specifications are made for the same characters. If a TextRange object only uses one font, then the return value is True (non-zero) or False (0).

Example The following example creates a shape in the active diagram, and adds some text to the shape. The shape’s TextRange object is retrieved through the TextBlock object, and the entire text string is assigned. Then the font size of the shape’s text is set to 16 points.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxTextRange As TextRange
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape1.Text = "Some shape text"
MsgBox "View the shape before changing the font characteristics"
' Get the shape’s TextRange object
Set igxTextRange = igxShape1.TextBlock.TextRange
igxTextRange.Font.Size = 16
MsgBox "View the results"
```

{button Font object,JI(`igrafxrf.HLP',`Font_Object')}

Strikethrough Property

Syntax *Font*.**Strikethrough**

Data Type Long (read/write)

Description The Strikethrough property specifies whether to use a strikethrough line for the Font object. A value of zero equals False; any non-zero value equals True.

In the case of a TextRange, which can use more than one font, this property can return the value of ixUndefined (99999999) if conflicting font specifications are made for the same characters. If a TextRange object only uses one font, then the return value is True (non-zero) or False (0).

Example The following example creates a shape in the active diagram, and adds some text to the shape. The shape's TextRange object is retrieved through the TextBlock object, and the entire text string is assigned. Then the Strikethrough property is set to True, causing all text for the shape to have a strikethrough line.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxTextRange As TextRange
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape1.Text = "Some shape text"
MsgBox "View the shape before changing the font characteristics"
' Get the shape's TextRange object
Set igxTextRange = igxShape1.TextBlock.TextRange
igxTextRange.Font.Strikethrough = 1
MsgBox "View the results"
```

{button Font object,JI(`igrafxrf.HLP',`Font_Object')}

Underline Property

Syntax *Font*.**Underline**

Data Type Long (read/write)

Description The Underline property specifies whether to underline the Font object. A single underline is the only option. A value of zero equals False; any non-zero value equals True.

In the case of a TextRange, which can use more than one font, this property can return the value of ixUndefined (99999999) if conflicting font specifications are made for the same characters. If a TextRange object only uses one font, then the return value is True (non-zero) or False (0).

Example The following example creates a shape in the active diagram, and adds some text to the shape. The shape's TextRange object is retrieved through the TextBlock object, and the entire text string is assigned. Then the Underline property is set to True so all the shape's text is underlined.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxTextRange As TextRange
Dim iCount As Integer
' Create a shape in the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
igxShape1.Text = "Some shape text"
MsgBox "View the shape before changing the font characteristics"
' Get the shape's TextRange object
Set igxTextRange = igxShape1.TextBlock.TextRange
igxTextRange.Font.Underline = 1
MsgBox "View the results"
```

{button Font object,JI(`igrafxrf.HLP',`Font_Object')}

FontNames Object

The FontNames object is a collection that contains the names, as strings, of all the fonts that are installed on the system. This object is only accessible from the Application object.

The FontNames object provides the following functionality:

- The ability to access the name of any font that is installed on the system.
- The ability to determine how many fonts are currently in the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the FontNames object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
Parent		

Related Topics

Font object

Item Method

Syntax *FontNames*.**Item**(*Index* As Integer) As String

Description The Item method returns the name of the font at the specified *Index* from the FontNames collection. The data type of the *Index* argument is Integer. The result of the method is a string (the Font name), and must be assigned to a String variable. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example lists all the fonts in the FontNames collection to the Output window. These are the fonts available for use within iGrafx Professional.

```
' Dimension the variables
Dim igxFonts As FontNames
Dim sFonts As String
Dim iCount, Remainder As Integer
sFonts = ""
' Get the application's FontNames collection
Set igxFonts = Application.FontNames
' Collect the list of font names in a string
For iCount = 1 To igxFonts.Count
    Remainder = iCount Mod 3
    If (Remainder = 0) Then
        Output sFonts & igxFonts.Item(iCount)
        sFonts = ""
    Else
        sFonts = sFonts & igxFonts.Item(iCount) & ", "
    End If
Next iCount
```

{button FontNames object,JI(`igrafxrf.HLP',`FontNames_Object')}

Grid Object

The Grid object controls the display of the grid (sometimes called a construction, or “snap to” grid) for the application. The Grid object is subordinate to, and accessed from, the Application object; therefore, the grid is either on or off for all diagrams within the application.

With this object, a developer can control whether a grid is displayed, whether the grid SnapTo function is active, and what the settings of the horizontal and vertical grid spacings. The properties available through the API provide the same control over the grid as is available through the iGrafx Professional user interface.

Properties, Methods, and Events

All of the properties, methods, and events for the Grid object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	SnapPoint	
GridSpacing		
HorizontalSpacing		
Parent		
SnapToGrid		
VerticalSpacing		
Visible		

Related Topics

Application.Grid property

GridSpacing Property

Syntax *Grid*.**GridSpacing**

Data Type IxSnap enumerated constant (read/write)

Description The GridSpacing property specifies the spacing for both the X and Y directions of the grid. Use this property to set grid spacing to a predefined amount specified by an IxSnap constant. To vary the grid spacing in the X and Y directions, or to set the grid spacing to an amount other than those available with the IxSnap constants, use the HorizontalSpacing and VerticalSpacing properties.

The IxSnap constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant	Grid Spacing
0	ixSnapCoarseEnglish	1/8 inch
1	ixSnapMediumEnglish	1/12 inch
2	ixSnapFineEnglish	1/16 inch
3	ixSnapCoarseMetric	½ inch
4	ixSnapMediumMetric	1/5 inch
5	ixSnapFineMetric	1/10 inch

Example The folowing example makes the grid visible, and then sets the grid spacing to each of the IxSnap constants.

```
' Dimension the variables
Dim igxGrid As Grid
Dim iCount As Integer
' Get the application's Grid and make it visible
Set igxGrid = Application.Grid
igxGrid.Visible = True
' Set the grid spacing to each of the IxSnap constants
For iCount = 0 To 5
    igxGrid.GridSpacing = iCount
    MsgBox "View the grid spacing"
Next iCount
```

See Also HorizontalSpacing property

VerticalSpacing property

{button Grid object,JI(`igrafxrf.HLP',`Grid_Object')}

HorizontalSpacing Property

Syntax *Grid*.**HorizontalSpacing**

Data Type Long (read/write)

Description The HorizontalSpacing property specifies the horizontal (X direction) spacing of the grid. The units for this property are twips (1440 twips = 1 inch). Using this property and the VerticalSpacing property, you can set grid spacing to unequal amounts in either direction. To set the spacing in both directions at once equally, use the GridSpacing property.

Example The following example displays the grid and sets the horizontal spacing to 1 inch and the vertical spacing to 1/4 inch.

```
' Dimension the variables
Dim igxGrid As Grid
Dim iCount As Integer
' Get the application's Grid and make it visible
Set igxGrid = Application.Grid
igxGrid.Visible = True
' Set the horizontal grid spacing to 1 inch and vertical to 1/4 inch
igxGrid.HorizontalSpacing = 1440
igxGrid.VerticalSpacing = 360
MsgBox "View the grid spacing"
```

See Also VerticalSpacing property

GridSpacing property

{button Grid object,JI(`igrafxrf.HLP',`Grid_Object')}

SnapPoint Method

Syntax *Grid*.**SnapPoint**(*X* As Long, *Y* As Long, *SnappedX* As Long, *SnappedY* As Long)

Description The SnapPoint method snaps the point specified by the *X* and *Y* arguments, in units of twips, to the grid. The resulting snapped point is returned as SnappedX and Snapped*Y*, in twips.

Create Long variables and insert them in at the SnappedX and Y positions

Example The following example

{button Grid object,JI(`igrafxrf.HLP',`Grid_Object')}

SnapToGrid Property

Syntax *Grid*.**SnapToGrid**[= {**True | False**}]

Data Type Boolean (read/write)

Description The SnapToGrid property specifies whether objects you add to the diagram are snapped to the grid. Setting this property to True is the equivalent of turning on the Snap To Grid option in the Arrange—Grid menu.

{button Grid object,JI(`igrafxrf.HLP',`Grid_Object')}

VerticalSpacing Property

Syntax *Grid*.**VerticalSpacing**

Data Type Long (read/write)

Description The VerticalSpacing property specifies the vertical (Y direction) spacing of the grid. The units for this property are twips (1440 twips = 1 inch). Using this property and the HorizontalSpacing property, you can set grid spacing to unequal amounts in either direction. To set the spacing in both directions at once equally, use the GridSpacing property.

Example The following example displays the grid and sets the horizontal spacing to 1 inch and the vertical spacing to 1/4 inch.

```
' Dimension the variables
Dim igxGrid As Grid
Dim iCount As Integer
' Get the application's Grid and make it visible
Set igxGrid = Application.Grid
igxGrid.Visible = True
' Set the horizontal grid spacing to 1 inch and vertical to 1/4 inch
igxGrid.HorizontalSpacing = 1440
igxGrid.VerticalSpacing = 360
MsgBox "View the grid spacing"
```

See Also HorizontalSpacing property

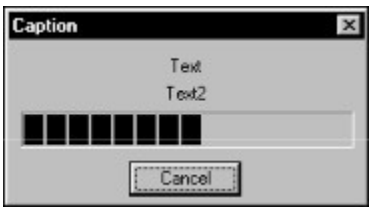
GridSpacing property

{button Grid object,JI(`igrafxrf.HLP',`Grid_Object')}

PercentGauge Object

The PercentGauge object controls the display of a percent gauge user interface tool. Percent gauges typically are used to provide users with visual feedback regarding task completion, such as file downloads, installation setups, etc.

The following illustration shows the standard percent gauge form, and the locations of the Caption, Text, and Text2 properties.



The following example gets the PercentGauge object from the Application object and then sets all of the properties and tests the value property.

```
' Dimension the variables
Dim igxPGauge As PercentGauge
Dim iCount As Long
' Get the PercentGauge object from the application
Set igxPGauge = Application.PercentGauge
' Set the CanceledEnabled property to True
igxPGauge.CancelEnabled = True
' Set the properties of the percent gauge object
igxPGauge.Caption = "PercentGauge Test"
igxPGauge.Text = "This is the top text."
igxPGauge.Text2 = "This is the bottom text."
igxPGauge.Visible = True
' Create values and set them into the value property of
' the percent gauge
For iCount = 1 To 100000
    igxPGauge.Value = Int(iCount / 1000)
    If (Canceled) Then
        Exit For
    End If
Next iCount
' Hide the percent gauge when done
igxPGauge.Visible = False
```

Properties, Methods, and Events

All of the properties, methods, and events for the PercentGauge object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Canceled		
CancelEnabled		
Caption		

Parent
Text
Text2
Value
Visible

Related Topics

Application.PercentGauge property

Canceled Property

Syntax *PercentGauge*.**Canceled**[= {**True | False**}]

Data Type Boolean (read-only)

Description The Canceled property indicates whether the user has pressed the Cancel button on the percent gauge form. The developer should occasionally check this property during an operation to determine whether the user has canceled it.

For an example that uses this property, refer to the discussion of the PercentGauge object.

See Also CancelEnabled property

{button PercentGauge object,JI(`igrafxrf.HLP',`PercentGauge_Object')}

CancelEnabled Property

Syntax *PercentGauge*.**CancelEnabled**[= {**True | False**}]

Data Type Boolean (read/write)

Description The CancelEnabled property specifies whether or not the Cancel button is enabled in the percent gauge. By default, the Cancel button is enabled; therefore, enabling the cancel is not necessary during initialization of the percent gauge. With Cancel enabled, it allows the user to cancel the dialog (and therefore, the action).

If the user cancels the operation the percent gauge in monitoring, the Canceled property returns True.

You should be sure that there is a good reason to disable the Cancel button for the percent gauge. It is possible that the application can get into a state where it is frozen; without the Cancel button, the user would have no means of attempting to re-establish control.

For an example that uses this property, refer to the discussion of the PercentGauge object.

See Also Canceled property

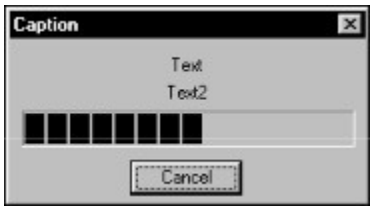
{button PercentGauge object,JI(`igrafxrf.HLP',`PercentGauge_Object')}

Text2 Property

Syntax *PercentGauge*.**Text2**

Data Type String (read/write)

Description The Text2 property specifies the text that is displayed in the Text2 area (the second line) of the PercentGauge object, as shown in the following illustration.



The Text property specifies the first line of text.
For an example that uses this property, refer to the discussion of the PercentGauge object.

See Also Text property

{button PercentGauge object,JI(`igrafxrf.HLP',`PercentGauge_Object')}

Value Property

Syntax *PercentGauge*.**Value**

Data Type Integer (read/write)

Description The Value property specifies the value of the percent gauge that is displayed in the progress bar. Valid values for the percent gauge are 0 through 100.

For an example that uses this property, refer to the discussion of the PercentGauge object.

{button PercentGauge object,JI(`igrafxrf.HLP',`PercentGauge_Object')}

RecentFiles Object

The RecentFiles object is a collection that contains the names, as strings, of the most recently opened files. This collection corresponds to the list of recently opened files in the File menu. The RecentFiles collection is only associated with and accessed from the Application object. The number of file names displayed is controlled by the application according to specific settings for the maximum number of files.

The RecentFiles object provides the following functionality:

- The ability to access the name of any filename string in the collection.
- The ability to determine how many items are currently in the collection.
- The ability to set the maximum number of filenames that can be contained in the collection, and therefore, displayed in the File menu.
- The ability to add a new string to the collection.
- The ability to open a file whose name is in the collection.
- The ability to remove a filename from the collection.
- The ability to change the order of the filenames in the collection by moving them to the top or bottom of the list.

The RecentFiles collection is populated automatically as files are opened, or when new files are saved. You can also add to the collection programmatically without ever opening the file, as well as open files, remove files, and change the order of the files in the collection.

Note that new additions to the collection are added to the beginning, not to the end. For example, if File1 is opened, then File2 is opened, File 2 is at Index 1 in the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the RecentFiles object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Maximum	MoveToBottom	
Parent	MoveToTop	
	Open	
	Remove	

Related Topics

Application.RecentFiles property

Add Method

Syntax *RecentFiles*.**Add** (*Name* As String)

Description The Add method adds a filename to the RecentFiles collection. The *Name* argument specifies the name of the file. You should specify the full path to the file.

The method does not check whether the filename you specify with the *Name* argument exists, or is a valid type for use with iGrafx Professional. The *Name* argument is just a string, and the RecentFiles collection will accept any string value it is given. The validity or existence of the file is tested when you or a user try to open it.

Example The following example gets the application's RecentFiles collection and tests whether it contains any entries. It then adds a file to the collection. Check the recent files list in the file menu before and after running this code.

```
' Dimension the variables
Dim igxRecentFiles As RecentFiles
Dim sFiles As String
Dim iCount As Integer
' Get the application's RecentFiles collection
Set igxRecentFiles = Application.RecentFiles
If (igxRecentFiles.Count > 0) Then
   sFiles = ""
   For iCount = 1 To igxRecentFiles.Count
       sFiles = sFiles & igxRecentFiles.Item(iCount) & Chr(13)
   Next iCount
   MsgBox "The Recent Files collection contains the " _
       & "following files: " & Chr(13) & sFiles
Else
   MsgBox "RecentFiles collection is empty."
End If
Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
    & "Pro\8.0\Exercise\Ex6.igx")
If (igxRecentFiles.Count > 0) Then
   sFiles = ""
   For iCount = 1 To igxRecentFiles.Count
       sFiles = sFiles & igxRecentFiles.Item(iCount) & Chr(13)
   Next iCount
   MsgBox "The Recent Files collection contains the " _
       & "following files: " & Chr(13) & sFiles
Else
   MsgBox "RecentFiles collection is empty."
End If
```

{button RecentFiles object,JI(`igrafxrf.HLP',`RecentFiles_Object')}

Item Method

Syntax *RecentFiles*.**Item**(*Index* As Integer) As String

Description The Item method returns the filename at the specified *Index* from the RecentFiles collection. The data type of the *Index* argument is Integer. The result of the method is a string (the filename), and must be assigned to a String variable. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example lists all the files in the RecentFiles collection in a message box. Notice that only the last four files are listed. This is because the Maximum number of files allowed in the collection is four.

```
' Dimension the variables
Dim igxRecentFiles As RecentFiles
Dim sFiles As String
Dim iCount As Integer
' Get the application's RecentFiles collection
Set igxRecentFiles = Application.RecentFiles
' Add 8 files to the collection
For iCount = 1 To 8
    Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
        & "Pro\8.0\Exercise\Ex" & iCount & ".igx")
Next iCount
' List the files in the collection in a message box
If (igxRecentFiles.Count > 0) Then
    sFiles = ""
    For iCount = 1 To igxRecentFiles.Count
        sFiles = sFiles & igxRecentFiles.Item(iCount) & Chr(13)
    Next iCount
    MsgBox "The Recent Files collection contains the " _
            & "following files: " & Chr(13) & sFiles
Else
    MsgBox "RecentFiles collection is empty."
End If
```

Maximum Property

Syntax *RecentFiles*.**Maximum**

Data Type Integer (read-only)

Description The Maximum property returns the maximum number of files in the RecentFiles collection. This value is defined to be 4 by the iGrafx Professional application, and cannot be changed.

Example The following example prints the value of the Maximum property in a message box.

```
' Dimension the variables
Dim igxRecentFiles As RecentFiles
Dim sFiles As String
Dim iCount, MaxFiles As Integer
' Get the application's RecentFiles collection
Set igxRecentFiles = Application.RecentFiles
MaxFiles = igxRecentFiles.Maximum
' Print the maximum number of files the collection can contain
MsgBox "The RecentFiles collection can contain a maximum" _
    & Chr(13) & "of " & MaxFiles & " files."
' Add 8 files to the collection
```

{button RecentFiles object,JI(`igrafxrf.HLP',`RecentFiles_Object')}

MoveToBottom Method

Syntax *RecentFiles*.**MoveToBottom (***Index* As Integer)

Description The MoveToBottom method moves the filename at the specified *Index* within the RecentFiles collection to the end of the RecentFiles collection (and consequently, to the bottom of the list of recently opened files in the File menu). For example, if you move the second file in the list to the end, the third and fourth files move up in the list, becoming the second and third files.

An error is returned if the index is invalid. Use the Maximum property to prevent invalid index errors.

Error If you specify an invalid index value, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example fills the RecentFiles collection with four files, and prints all the filenames to the Output window. Then the second file is moved to the bottom of the list. The filesnames are again printed to the Output window.

```
' Dimension the variables
Dim igxRecentFiles As RecentFiles
Dim iCount, MaxFiles As Integer
' Get the application's RecentFiles collection
Set igxRecentFiles = Application.RecentFiles
MaxFiles = igxRecentFiles.Maximum
' Add the maximum number of files to the collection
For iCount = 1 To MaxFiles
    Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
        & "Pro\8.0\Exercise\Ex" & iCount & ".igx")
Next iCount
' List the files in the collection in the Output window
If (igxRecentFiles.Count > 0) Then
    For iCount = 1 To igxRecentFiles.Count
        Output igxRecentFiles.Item(iCount)
    Next iCount
Else
    MsgBox "RecentFiles collection is empty."
End If
' Move the second file to the bottom of the list and output
' the list again
Call igxRecentFiles.MoveToBottom(2)
If (igxRecentFiles.Count > 0) Then
    For iCount = 1 To igxRecentFiles.Count
        Output igxRecentFiles.Item(iCount)
    Next iCount
Else
    MsgBox "RecentFiles collection is empty."
End If
```

See Also MoveToTop method

{button RecentFiles object,JI(`igrafxrf.HLP',`RecentFiles_Object')}

MoveToTop Method

Syntax *RecentFiles*.**MoveToTop** (*Index* As Integer)

Description The MoveToTop method moves the filename at the specified *Index* within the RecentFiles collection to the beginning of the RecentFiles collection (and consequently, to the top of the list of recently opened files in the File menu). For example, if you move the third file in the list to the beginning, the first and second files move down in the list, becoming the second and third files.

An error is returned if the index is invalid. Use the Maximum property to prevent invalid index errors.

Error If you specify an invalid index value, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example fills the RecentFiles collection with four files, and prints all the filenames to the Output window. Then the third file is moved to the top of the list. The filesnames are again printed to the Output window.

```
' Dimension the variables
Dim igxRecentFiles As RecentFiles
Dim sFiles As String
Dim iCount, MaxFiles As Integer
' Get the application's RecentFiles collection
Set igxRecentFiles = Application.RecentFiles
MaxFiles = igxRecentFiles.Maximum
' Add the maximum number of files to the collection
For iCount = 1 To MaxFiles
    Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
        & "Pro\8.0\Exercise\Ex" & iCount & ".igx")
Next iCount
' List the files in the collection in the Output window
If (igxRecentFiles.Count > 0) Then
    For iCount = 1 To igxRecentFiles.Count
        Output igxRecentFiles.Item(iCount)
    Next iCount
Else
    MsgBox "RecentFiles collection is empty."
End If
' Move the third file to the top of the list and output
' the list again
Call igxRecentFiles.MoveToTop(3)
If (igxRecentFiles.Count > 0) Then
    For iCount = 1 To igxRecentFiles.Count
        Output igxRecentFiles.Item(iCount)
    Next iCount
Else
    MsgBox "RecentFiles collection is empty."
End If
```

See Also MoveToBottom method

{button RecentFiles object,JI(`igrafxrf.HLP',`RecentFiles_Object')}

Open Method

Syntax *RecentFiles*.**Open**(*Index* As Integer)

Description The Open method opens the filename at the specified *Index* within the RecentFiles collection. An error is returned if the index is invalid, or if the file does not exist or is not a valid file type for iGrafx Professional. You can use the Maximum property to prevent invalid index errors.

Error If you specify an invalid index value, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example places three of the iGrafx Professional example files in the RecentFiles collection, and then opens the file located at Index 2.

```
' Dimension the variables
Dim igxRecentFiles As RecentFiles
Dim sFiles As String
Dim iCount As Integer
' Get the application's RecentFiles collection
Set igxRecentFiles = Application.RecentFiles
' Add three of the example files to the collection
Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
    & "Pro\8.0\Exercise\Ex6.igx")
Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
    & "Pro\8.0\Exercise\Ex7.igx")
Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
    & "Pro\8.0\Exercise\Ex8.igx")
' List the files in the collection in the Output window
If (igxRecentFiles.Count > 0) Then
    For iCount = 1 To igxRecentFiles.Count
        Output igxRecentFiles.Item(iCount)
    Next iCount
Else
    MsgBox "RecentFiles collection is empty."
End If
' Open the second file in the list
Call igxRecentFiles.Open(2)
```

{button RecentFiles object,JI(`igrafxrf.HLP',`RecentFiles_Object')}

Remove Method

Syntax *RecentFiles*.**Remove** (*Index* As Integer)

Description The Remove method removes the filename at the specified *Index* within the RecentFiles collection. An error is returned if the index is invalid. Use the Maximum property to prevent invalid index errors.

Error If you specify an invalid index value, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example places three of the iGrafx Professional example files in the RecentFiles collection. It prints the list of files to the Output window, removes the file located at Index 2, and then prints the list again.

```
' Dimension the variables
Dim igxRecentFiles As RecentFiles
Dim sFiles As String
Dim iCount As Integer
' Get the application's RecentFiles collection
Set igxRecentFiles = Application.RecentFiles
' Add three of the example files to the collection
Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
    & "Pro\8.0\Exercise\Ex6.igx")
Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
    & "Pro\8.0\Exercise\Ex7.igx")
Call igxRecentFiles.Add("C:\Program Files\iGrafx\" _
    & "Pro\8.0\Exercise\Ex8.igx")
' List the files in the collection in the Output window
If (igxRecentFiles.Count > 0) Then
    For iCount = 1 To igxRecentFiles.Count
        Output igxRecentFiles.Item(iCount)
    Next iCount
Else
    MsgBox "RecentFiles collection is empty."
End If
' Remove the second file in the list
Call igxRecentFiles.Remove(2)
' List the files in the collection in the Output window
If (igxRecentFiles.Count > 0) Then
    For iCount = 1 To igxRecentFiles.Count
        Output igxRecentFiles.Item(iCount)
    Next iCount
Else
    MsgBox "RecentFiles collection is empty."
End If
```

{button RecentFiles object,JI(`igrafxrf.HLP',`RecentFiles_Object')}

Ruler Object

The Ruler object controls whether the ruler bar is visible and the units it displays. The Ruler object is only associated with and accessible from the Application object; therefore, if the Ruler is visible, it is visible within all iGrafx Professional diagram windows.

Properties, Methods, and Events

All of the properties, methods, and events for the Ruler object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Parent		
Units		
Visible		

Related Topics

Application.Ruler property

Units Property

Syntax *Ruler*.**Units**

Data Type IxUnits enumerated constant (read/write)

Description The Units property determines which units are displayed by the ruler bar, inches or centimeters. Even though the units are represented in the ruler as either centimeters or inches, the drawing area is still measured in twips.

The IxUnits constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	fcUnitsInches
1	fcUnitsCentimeters

Example The following example toggles the Units property based on its current setting.

```
' Dimension the variables
Dim igxRuler As Ruler
' Get the Ruler object and store it in the Ruler variable
Set igxRuler = Application.Ruler
MsgBox "View the ruler units."
' Toggle the units property depending on its current setting
If (igxRuler.Units = ixUnitsInches) Then
   ' Change the ruler to centimeters
   igxRuler.Units = ixUnitsCentimeters
   MsgBox "View the change to the ruler."
End If
If (igxRuler.Units = ixUnitsCentimeters) Then
   ' Change the ruler to inches
   igxRuler.Units = ixUnitsInches
   MsgBox "View the change to the ruler."
End If
```

See Also Application.ActiveUnits property

{button Ruler object,JI(`igrafxrf.HLP',`Ruler_Object')}

StatusBar Object

The StatusBar object represents the status bar at the bottom of the application window. The object controls whether the status bar is visible, and the text it displays. The StatusBar object is only associated with and accessible from the Application object. If the StatusBar is visible, it is visible for all iGrafx Professional windows.

The main purpose of the status bar is for displaying messages or some other information to a user. The status bar can display approximately 90 characters; however, the width of individual characters influences the exact number.

Properties, Methods, and Events

All of the properties, methods, and events for the StatusBar object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Parent		
Text		
Visible		

Related Topics

Application.StatusBar property

Application.Hint method

OutputPane Object

The OutputPane object represents an individual output pane in the Output Window. Output panes allow you to selectively display information in the Output window. For example, in the following illustration, the Output window has three output panes: Output, Trace, and Tools.

Properties, Methods, and Events

All of the properties, methods, and events for the OutputPane object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Activate	
Caption	AddString	
Parent	Clear	
	Delete	
	RemoveString	
	ReplaceString	

Related Topics

OutputPanes object

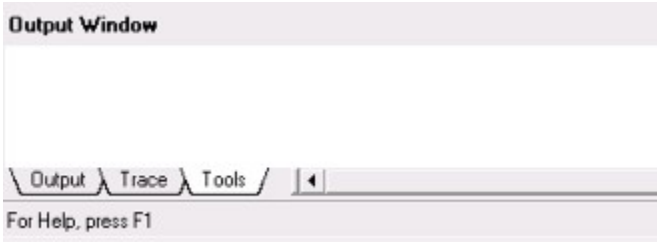
OutputWindow object

iGrafx API Object Hierarchy

Activate Method

Syntax *OutputPane*.**Activate**()

Description The Activate method activates the specified OutputPane object. In the following illustration, the Tools output pane is active.



Example The following example shows how to add three output panes to the interface, and then activate the second output pane.

```
' Dimension the variables
Dim igxOutWin As OutputWindow
Dim igxOutPanes As OutputPanes
Dim igxOutPane1 As OutputPane
Dim igxOutPane2 As OutputPane
Dim igxOutPane3 As OutputPane
' Get the OutputWindow from the Application object
Set igxOutWin = Application.OutputWindow
' Get the OutputPanes collection from the OutputWindow object
Set igxOutPanes = igxOutWin.OutputPanes
' Add three new output panes
Set igxOutPane1 = igxOutPanes.Add("Pane 1")
Set igxOutPane2 = igxOutPanes.Add("Pane 2")
Set igxOutPane3 = igxOutPanes.Add("Pane 3")
' Activate the second output pane
igxOutPane2.Activate
```

{button OutputPane object,JI(`igrafxrf.HLP',`OutputPane_Object')}

AddString Method

Syntax *OutputPane*.**AddString**(*String* As String) As Long

Description The AddString method adds a string to the end of the list in the specified output pane. The method returns a number that is a "key", which you can hold onto and use when handing the application's OutputWindowGoTo event. For many purposes, the Key should be stored in a global variable.

Note that the result of the method does not have to be assigned to a variable. In fact, all strings sent to an output pane have a key number; the issue is whether you want to be able to refer to that string later. Refer to the example for the Remove method.

The OutputWindowGoTo event fires when the user double clicks on a string in an output pane. The event has two parameters: *Key* as Long, and *Handled* As Boolean. If a string gets double clicked on, the Key value passed to the OutputWindowGoTo event matches the key returned by the AddString method. In this case, you should do whatever action is appropriate, such as selecting the shape or shapes that the output window message corresponds to, and set the *Handled* parameter of the event to True.

Example The following example adds three strings to an output pane named "Pane 1". It stores the keys that are returned in global variables. The keys are then listed in the "Output" pane of the Output window.

```
Dim Key1, Key2, Key3 As Long

Public Sub MyTest()

' Dimension the variables
Dim igxOutWin As OutputWindow
Dim igxOutPanes As OutputPanes
Dim igxOutPane As OutputPane
' Get the OutputWindow from the Application object
Set igxOutWin = Application.OutputWindow
' Get the OutputPanes collection from the OutputWindow object
Set igxOutPanes = igxOutWin.OutputPanes
' Add a new pane to the OutputPanes collection
Set igxOutPane = igxOutPanes.Add("Pane 1")
' Add the strings to the output pane
Key1 = igxOutPane.AddString("1st String")
Key2 = igxOutPane.AddString("2nd String")
Key3 = igxOutPane.AddString("3rd String")
Output "Three strings with keys added to Pane 1."
Output "The keys are: " & Str(Key1) & Str(Key2) & Str(Key3)

End Sub
```

See Also RemoveString method

ReplaceString method

OutputWindowGoTo event

{button OutputPane object,JI(`igrafxrf.HLP',`OutputPane_Object')}

Clear Method

Syntax *OutputPane*.**Clear**

Description The Clear method deletes the entire contents of the specified output pane.

Example The following example clears the contents of each output pane in the OutputPanes collection.

```
' Dimension the variables
Dim igxOutWin As OutputWindow
Dim igxOutPanes As OutputPanes
Dim igxOutPane As OutputPane
' Get the OutputWindow from the Application object
Set igxOutWin = Application.OutputWindow
' Get the OutputPanes collection from the OutputWindow object
Set igxOutPanes = igxOutWin.OutputPanes
' Go through the OutputPanes collection and
' clear the contents of each output pane
For Each igxOutPane In igxOutPanes
   igxOutPane.Clear
Next
```

{button OutputPane object,JI(`igrafxrf.HLP',`OutputPane_Object')}

RemoveString Method

Syntax *OutputPane*.**RemoveString** (*theKey* As Long)

Description The RemoveString method removes a string from the specified output pane. The string to remove is identified by the *theKey* argument, which is the key returned by the AddString method.

Note that the AddString method does not require assignment of the key to a variable (refer to the example). However, without a key, you cannot use the Remove or Replace methods, or the application's OutputWindowGoTo event. For more information on keys, see the AddString method.

Example The following example adds three strings to the output pane, and then removes the second string from the list.

```
' Dimension the variables
Dim igxOutWin As OutputWindow
Dim igxOutPanes As OutputPanes
Dim igxOutPane As OutputPane
Dim storedKey as Long
' Get the OutputWindow from the Application object
Set igxOutWin = Application.OutputWindow
' Get the OutputPanes collection from the OutputWindow object
Set igxOutPanes = igxOutWin.OutputPanes
' Add a new pane to the OutputPanes collection
Set igxOutPane = igxOutPanes.Add("Pane 1")
' Add the strings to the output pane
igxOutPane.AddString ("1st String")
storedKey = igxOutPane.AddString ("2nd String")
igxOutPane.AddString ("3rd String")
Application.OutputWindow.Visible = True
MsgBox "View the results"
' Remove the second string form the output pane
igxOutPane.RemoveString(storedKey)
MsgBox "Second string removed."
```

See Also AddString method

ReplaceString method

{button OutputPane object,JI(`igrafxrf.HLP',`OutputPane_Object')}

ReplaceString Method

Syntax *OutputPane*.**ReplaceString** (*theKey* As Long, *NewString* As String)

Description The ReplaceString method replaces a string in the specified output pane. The string to remove is identified by the *theKey* argument, which is the key returned by the AddString method. The *NewString* argument specifies the text of the replacement string.

Note that the AddString method does not require assignment of the key to a variable (refer to the example). However, without a key, you cannot use the Remove or Replace methods, or the application's OutputWindowGoTo event. For more information on keys, see the AddString method.

Example The following example adds three strings to the output pane, and then replaces the text of the second string with a new text string.

```
' Dimension the variables
Dim igxOutWin As OutputWindow
Dim igxOutPanes As OutputPanes
Dim igxOutPane As OutputPane
Dim storedKey as Long
' Get the OutputWindow from the Application object
Set igxOutWin = Application.OutputWindow
' Get the OutputPanes collection from the OutputWindow object
Set igxOutPanes = igxOutWin.OutputPanes
' Add a new pane to the OutputPanes collection
Set igxOutPane = igxOutPanes.Add("Pane 1")
' Add the strings to the output pane
igxOutPane.AddString ("1st String")
storedKey = igxOutPane.AddString ("2nd String")
igxOutPane.AddString ("3rd String")
Application.OutputWindow.Visible = True
MsgBox "View the results"
' Replace the second string form the output pane
igxOutPane.ReplaceString storedKey, "New SecondString"
MsgBox "Second string replaced."
```

See Also AddString method

RemoveString method

{button OutputPane object,JI(`igrafxrf.HLP',`OutputPane_Object')}

OutputPanes Object

The OutputPanes object is a collection of individual OutputPane objects. An OutputPanes collection is only associated with and accessible from the Application object. Its purpose is to store and provide access to the individual OutputPane objects that have been created in the OutputWindow object.

In the following illustration, there are three panes in the output window.

The OutputPanes object provides the following functionality:

- The ability to access any OutputPane objects that have been created.
- The ability to determine how many OutputPane objects are currently in the collection.
- The ability to add a new OutputPane object to the OutputPanes collection, and therefore, to the application's output window.

Properties, Methods, and Events

All of the properties, methods, and events for the OutputPanes object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent		

Related Topics

OutputPane object

OutputWindow object

iGrafx API Object Hierarchy

Add Method

Syntax *OutputPanes*.**Add** (*Caption* As String) As OutputPane

Description The Add method adds an OutputPane object to the OutputPanes collection. The *Caption* argument specifies the name of the new output pane. The caption is displayed as a Tab at the bottom left of the OutputWindow. The result of the method is an OutputPane object, and must be assigned to a variable of type OutputPane. The newly created pane becomes the active pane.

It is important to note that the OutputPanes collection may be empty initially, and that the OutputWindow may be hidden.

Example The following example adds a new output pane named "MyPane" to the OutputPanes collection of the OutputWindow object. This pane becomes the active pane in the output window.

```
' Dimension the variables
Dim igxOutWin As OutputWindow
Dim igxOutPane As OutputPane
' Get the OutputWindow from the Application object
Set igxOutWin = Application.OutputWindow
' Make the Output Window visible
igxOutWin.Visible = True
' Add an OutputPane to the OutputPanes collection
Set igxOutPane = igxOutWin.OutputPanes.Add("MyPane")
MsgBox "View the results"
```

To add a second pane, use the Add method again and specify a new caption name. This second output pane becomes the active pane.

```
' Add a second OutputPane to the OutputPanes collection
Set igxOutPane = igxOutWin.OutputPanes.Add("MyPane Number 2")
MsgBox "View the results"
```

To make the first pane the active pane again, assign it to the igxOutPane variable using the Item method, and then use the Activate method. The output pane titled "MyPane" now becomes the active pane.

```
Set igxOutPane = igxOutWin.OutputPanes.Item(1)
igxOutPane.Activate
MsgBox "View the results"
```

{button OutputPanes object,JI(`igrafxrf.HLP',`OutputPanes_Object')}

Item Method

Syntax *OutputPanes*.**Item**(*Index* As Integer) As OutputPane

Description The Item method returns the OutputPane object at the specified *Index* from the OutputPanes collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type OutputPane. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example creates 5 new output panes for the Output window. As each pane is added, a string is placed in the pane. Then each pane is activated in turn, and its caption is printed to the Output window using the application's Output method.

```
' Dimension the variables
Dim igxOutWin As OutputWindow
Dim igxOutPanes As OutputPanes
Dim igxOutPane As OutputPane
Dim storedKey As Long
' Get the OutputWindow from the Application object
Set igxOutWin = Application.OutputWindow
' Get the OutputPanes collection from the OutputWindow object
Set igxOutPanes = igxOutWin.OutputPanes
' Make the Output window visible
Application.OutputWindow.Visible = True
' Create 5 new output panes
For iCount = 1 To 5
   Set igxOutPane = igxOutPanes.Add("Pane" & Str(iCount))
   igxOutPane.AddString ("String " & Str(iCount))
   MsgBox "View the results."
Next iCount
' Change the Cpation property of the third output pane
igxOutPanes.Item(3).Caption = "Diagnostics"
' Activate each output pane and print the caption with the
' Output method
For iCount = 1 To igxOutPanes.Count
   Call igxOutPanes.Item(iCount).Activate
   MsgBox "View the results."
   Output "Output pane caption is: " _
       & igxOutPanes.Item(iCount).Caption
Next iCount
```

{button OutputPanes object,JI(`igrafxrf.HLP',`OutputPanes_Object')}

OutputWindow Object

The OutputWindow object represents the window that is shown when Output is selected from the View menu. This window is very useful when many panels of list data need to be displayed. The output window may have 0 or more output panes. The following illustration shows an output window that has three output panes: Output, Trace, and Tools.

Before you begin to manipulate the OutputWindow, you must first get the OutputWindow object from the Application object. The following example code shows how to get the OutputWindow object from the Application object.

```
' Dimension the variables
Dim igxOutWin As OutputWindow
' Get the OutputWindow from the Application object
Set igxOutWin = Application.OutputWindow
```

Once you get the OutputWindow object, you can control whether or not it is visible, and access the OutputPanes collection. Note that the OutputWindow object does not do anything without at least one output pane.

Properties, Methods, and Events

All of the properties, methods, and events for the OutputWindow object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
OutputPanes		
Parent		
Visible		

Related Topics

OutputPane object

OutputPanes object

iGrafx API Object Hierarchy

OutputPanes Property

Syntax *OutputWindow*.**OutputPanes**

Data Type OutputPanes collection object (read-only, See Object Properties)

Description The OutputPanes property returns the OutputPanes collection for the specified OutputWindow object. An output pane is an area where text can be placed using the Application.Output method, or the OutputPane.AddString method.

It is important to note that the OutputPanes collection may contain zero output panes.

Example The following example adds an OutputPane object to the OutputPanes collection of the OutputWindow object.

```
' Dimension the variables
Dim igxOutWin As OutputWindow
Dim igxOutPane As OutputPane
' Get the OutputWindow from the Application object
Set igxOutWin = Application.OutputWindow
' Add an OutputPane to the OutputPanes collection
Set igxOutPane = igxOutWin.OutputPanes.Add("MyPane")
```

See Also OutputPane object

OutputPanes object

iGrafx API Object Hierarchy

{button OutputWindow object,JI(`igrafxrf.HLP',`OutputWindow_Object')}

Window Object

The Window object is the representation of a window within the iGrafx Professional application. A Window object is associated with and accessible from the following objects:

- Application
- View

In addition, a Window object can be accessed from the Application and Document object through their Windows collection.

The Window object allows the developer to activate a window, close a window, set a window's position and size, give it a name, etc.

Properties, Methods, and Events

All of the properties, methods, and events for the Window object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Active	Activate	
Application	Center	
Caption	Close	
Handle	Flash	
Height		
Left		
Parent		
Top		
Visible		
Width		
WindowState		

Related Topics

PopupWindow object

Windows object

View object

DiagramView object

iGrafx API Object Hierarchy

Activate Method

Syntax *Window*.**Activate**

Description The Activate method activates the specified Window object. Activating a window means that it has the focus; that is, events (mouse clicks, keystrokes, etc.) are associated with or received by the window that currently has the focus.

Example The following example begins by switching the window state of the application window and then the document window. It then adds a new diagram view, which creates a new window. The Activate method is used to make the new diagram view window the active window. This is verified by using the Active property to display which window is currently active.

```
' Dimension the variables
Dim igxWindow As Window
Dim igxWindows As Windows
Dim igxShape As Shape
Dim igxDiagView As DiagramView
' Place a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the Application object's Window object
Set igxWindow = Application.Window
' Minimize and maximize the application window
MsgBox "Click OK to minimize the Application window."
igxWindow.WindowState = ixWindowMinimized
MsgBox "Click OK to maximize the Application window."
igxWindow.WindowState = ixWindowMaximized
' Get the first item in the Windows collection
' of the active document
Set igxWindow = ActiveDocument.Windows.Item(1)
' Minimize and maximize the document window
MsgBox "Click OK to minimize the Document window."
igxWindow.WindowState = ixWindowMinimized
MsgBox "Click OK to make the Document window Normal."
igxWindow.WindowState = ixWindowNormal
MsgBox "View the results."
MsgBox "Application Windows collection contains " _
   & Application.Windows.Count & " items."
' Create a view window on the current diagram
Set igxDiagView = ActiveDocument.Views.AddDiagramView(ActiveDiagram)
MsgBox "Application Windows collection contains " _
   & Application.Windows.Count & " items."
' Activate the new view window
Call igxDiagView.View.Window.Activate
Application.RefreshUI
MsgBox "View the results."
For Each igxWindow In Application.Windows
   If (igxWindow.Active) Then
       MsgBox igxWindow.Caption & " is the active window"
   Else
       MsgBox igxWindow.Caption & " is NOT the active window"
   End If
Next
```

See Also Active property

{button Window object,JI(`igrafxrf.HLP',`Window_Object')}

Active Property

Syntax *Window*.**Active**[= {**True | False**}]

Data Type Boolean (read-only)

Description The Active property indicates whether the specified Window object is the window with the focus (is the active window). A value of True means the window is active. This property is read-only, so you can only query whether the window is active. To activate a particular window, use the Activate method (this sets the value of this property to True).

Example The following example begins by switching the window state of the application window and then the document window. It then adds a new diagram view, which creates a new window. The Activate method is used to make the new diagram view window the active window. This is verified by using the Active property to display which window is currently active.

```
' Dimension the variables
Dim igxWindow As Window
Dim igxWindows As Windows
Dim igxShape As Shape
Dim igxDiagView As DiagramView
' Place a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the Application object's Window object
Set igxWindow = Application.Window
' Minimize and maximize the application window
MsgBox "Click OK to minimize the Application window."
igxWindow.WindowState = ixWindowMinimized
MsgBox "Click OK to maximize the Application window."
igxWindow.WindowState = ixWindowMaximized
' Get the first item in the Windows collection
' of the active document
Set igxWindow = ActiveDocument.Windows.Item(1)
' Minimize and maximize the document window
MsgBox "Click OK to minimize the Document window."
igxWindow.WindowState = ixWindowMinimized
MsgBox "Click OK to make the Document window Normal."
igxWindow.WindowState = ixWindowNormal
MsgBox "View the results."
MsgBox "Application Windows collection contains " _
   & Application.Windows.Count & " items."
' Create a view window on the current diagram
Set igxDiagView = ActiveDocument.Views.AddDiagramView(ActiveDiagram)
MsgBox "Application Windows collection contains " _
   & Application.Windows.Count & " items."
' Activate the new view window
Call igxDiagView.View.Window.Activate
Application.RefreshUI
MsgBox "View the results."
For Each igxWindow In Application.Windows
   If (igxWindow.Active) Then
       MsgBox igxWindow.Caption & " is the active window"
   Else
       MsgBox igxWindow.Caption & " is NOT the active window"
   End If
Next
```

See Also Activate method

{button Window object,JI(`igrafxrf.HLP',`Window_Object')}

Center Method

Syntax *Window*.**Center**

Description The Center method centers the specified Window object within the display screen.

Example The following example places a shape in the initial diagram window, then adds a second view window for the active diagram. It then uses the Center method to center both windows.

```
' Dimension the variables
Dim igxWindow As Window
Dim igxWindows As Windows
Dim igxShape As Shape
Dim igxDiagView As DiagramView
' Place a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the Application object's Window object
Set igxWindow = Application.Window
MsgBox "Application Windows collection contains " _
   & Application.Windows.Count & " items."
' Create a view window on the current diagram
Set igxDiagView = ActiveDocument.Views.AddDiagramView(ActiveDiagram)
MsgBox "Application Windows collection contains " _
   & Application.Windows.Count & " items."
' Center both windows in the application's Windows collection
For Each igxWindow In Application.Windows
   igxWindow.Center
   MsgBox "View the results."
Next
```

{button Window object,JI(`igrafxrf.HLP',`Window_Object')}

Close Method

Syntax *Window*.**Close**

Description The Close method closes the specified Window object.

Example The following example places a shape in the initial diagram window, then adds a second view window for the active diagram. It then uses the Close method to close the second window in the application's Windows collection.

```
' Dimension the variables
Dim igxWindow As Window
Dim igxWindows As Windows
Dim igxShape As Shape
Dim igxDiagView As DiagramView
' Place a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Get the Application object's Window object
Set igxWindow = Application.Window
MsgBox "Application Windows collection contains " _
   & Application.Windows.Count & " items."
' Create a view window on the current diagram
Set igxDiagView = ActiveDocument.Views.AddDiagramView(ActiveDiagram)
MsgBox "Application Windows collection contains " _
   & Application.Windows.Count & " items."
' Center both windows in the application's Windows collection
Set igxWindow = Application.Windows.Item(2)
igxWindow.Close
MsgBox "View the results."
```

{button Window object,JI(`igrafxrf.HLP',`Window_Object')}

Flash Method

Topic Under Construction!!!

Syntax *Window*.**Flash**

Description The Flash method toggles the colors of the specified Window object's title bar. You should use this method twice to toggle the window's title bar colors.

{button Window object,JI(`igrafxrf.HLP',`Window_Object')}

Handle Property

Syntax *Window*.**Handle**

Data Type Long (read-only)

Description The Handle property returns the specified Window object's handle. The handle can be used in Windows API functions that act on or require a window's handle as a parameter.

Window Handles are not normally used within Visual Basic. This property is provided to facilitate the use of Window Handles from within C++, and other programming languages that may require Window Handles for some tasks.

{button Window object,JI(`igrafxrf.HLP',`Window_Object')}

Left Property

Syntax *Window*.**Left**

Data Type Long (read/write)

Description The Left property specifies the location of the left side of a Window object. The units for this property are pixels, relative to the parent window. So a window off of a View would have a left that is relative to the parent window—the main application window.

{button Window object,JI(`igrafxrf.HLP',`Window_Object')}

WindowState Property

Syntax *Window*.**WindowState**

Data Type IxWindowState enumerated constant (read/write)

Description The WindowState property controls the state of the specified Window object. A window is always in one of the following states: Normal, Minimized, or Maximized.

The IxWindowState constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixWindowNormal
1	ixWindowMaximized
2	ixWindowMinimized

To control the state of all windows in the application, use the WindowState property of the Application object.

{button Window object,JI(`igrafxrf.HLP',`Window_Object')}

Windows Object

The Windows collection is a collection of individual Window objects. A Windows collection is associated with, and accessible from, the Application and Document objects. The Application object's Windows collection contains all MDI Child windows currently open in the application. A Document object's Windows collection contains all MDI Child windows associated with the document.

The Windows object provides the following functionality:

- The ability to access any Window object.
- The ability to determine how many Window objects are currently open.
- The ability to arrange how the windows are presented on screen to a user.

Properties, Methods, and Events

All of the properties, methods, and events for the Windows object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Arrange	
Count	Item	
Parent		

Related Topics

PopupWindow object

Window object

View object

DiagramView object

iGrafx API Object Hierarchy

Arrange Method

Syntax *Windows*.**Arrange**(*ArrangeType* As IxWindowsArrangeType)

Description The Arrange method arranges all the windows in the specified Windows collection.

The *ArrangeType* argument specifies how to arrange the windows on the screen. The IxWindowsArrangeType constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixArrangeIcons
1	ixCascade
2	ixTileHorizontal
3	ixTileVertical

{button Windows object,JI(`igrafxrf.HLP',`Windows_Object')}

Item Method

Syntax *Windows*.**Item**(*Index* As Integer) As Window

Description The Item method returns the Window object at the specified *Index* from the Windows collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Window. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example iterates through the Windows collection and prints to the Output window the caption of each Window object in the collection.

```
' Dimension the variables
Dim igxWindows As Windows
Dim iCount As Integer
Set igxWindows = Application.Windows
' Iterate through Windows collection and print
' the Window object's Caption
For iCount = 1 To igxWindows.Count
    Output igxWindows.Item(iCount).Caption
Next iCount
```

{button Windows object,JI(`igrafxrf.HLP',`Windows_Object')}

Workspace Object

The Workspace object provides functionality equivalent to File, Save Workspace… and File, Open where the file you open is a workspace file (*.igw;*.afw).

A Workspace file stores information about all the currently opened documents and windows. When you load the workspace file, it restores all the opened documents and windows.

Properties, Methods, and Events

All of the properties, methods, and events for the Workspace object are listed in the following table. Click the name to view the documentation for any particular property, method, or event.

Properties	Methods	Events
Application	Load	
Parent	Save	

Related Topics

Application.Workspace property

Load Method

Syntax *Workspace*.**Load**(*FileName* As String)

Description The Load method loads a previously saved workspace. The *Filename* argument specifies the name of the workspace file to load. The argument must be a valid workspace file (*.igw;*.afw), or an error is returned. It is recommended that you always supply a complete path name.

{button Workspace object,JI(`igrafxrf.HLP',`Workspace_Object')}

Save Method

Syntax *Workspace*.**Save**(*FileName* As String)

Description The Save method saves the current workspace. The *Filename* argument specifies the name under which to save the workspace file. It is recommended that you always supply a complete path name. Invoking this method displays the Save As dialog box.

Example The following example saves the current workspace as "MyNewWorkspace.igw".

```
Application.Workspace.Save "E:\MyNewWorkspace.igw"
```

{button Workspace object,JI(`igrafxrf.HLP',`Workspace_Object')}

View Object

Topic Under Construction!!!

The View object represents a view that displays the contents of a document, component, or diagram.

You can create custom views for a Document, Component, or Diagram object; therefore, a generic View object is used to provide for these "custom" views. For instance, the Tabular view is an example of a custom view. Using the Extensions architecture, you can create additional custom views.

If the View object is iGrafx Professional's built-in view (a DiagramView), then the View's DiagramView property returns that DiagramView object. If the View object is a custom view, then the View's DiagramView property returns the “Nothing” value.

Properties, Methods, and Events

All of the properties, methods, and events for the View object are listed in the following table. Click the name to view the documentation for any particular property, method, or event.

Properties	Methods	Events
Application	Close	
ClassID	Refresh	
DiagramView		
PageLayout		
Parent		
ProgID		
Window		

Related Topics

Window object

Views object

DiagramView object

iGrafx API Object Hierarchy

ClassID Property

Syntax *View*.**ClassID**

Data Type String (read-only)

Description The ClassID property returns a string that identifies the specified View object's class. This property, and the ProgID property, allow you to identify custom view types (the Tabular view is an example of a custom view type).

Example The following example

See Also ProgID property

{button View object,JI(`igrafxrf.HLP',`View_Object')}

Close Method

Syntax *View*.**Close**

Description The Close method closes the specified View object, and therefore, the window associated with the view.

{button View object,JI(`igrafxrf.HLP',`View_Object')}

DiagramView Property

Syntax *View*.**DiagramView**

Data Type DiagramView object (read-only, See Object Properties)

Description The DiagramView property returns the DiagramView object for the specified View object. If the specified View object is the iGrafx Professional built-in view (which is a DiagramView), then a DiagramView object is returned. The the specified View object is a custom view, this property returns ‘Nothing’.

See Also DiagramView object

iGrafx API Object Hierarchy

{button View object,JI(`igrafxrf.HLP',`View_Object')}

PageLayout Property

Syntax *View*.**PageLayout**

Data Type PageLayout object (read-only, See Object Properties)

Description The PageLayout property returns the PageLayout object associated with the specified View object.

See Also PageLayout object

iGrafx API Object Hierarchy

{button View object,JI(`igrafxrf.HLP',`View_Object')}

ProgID Property

Topic Under Construction!!!

Syntax *View*.**ProgID**

Data Type String (read-only)

Description

Example The following example

See Also ClassID property

{button View object,JI(`igrafxrf.HLP',`View_Object')}

Refresh Method

Topic Under Construction!!!

Syntax *View*.**Refresh**

Description The Refresh method forces the specified View object to refresh.

{button View object,JI(`igrafxrf.HLP',`View_Object')}

Window Property

Syntax *View*.**Window**

Data Type Window object (read-only, See Object Properties)

Description The Window property returns the Window object associated with the specified View object. The Window object provides additional properties and methods for positioning the window associated with the view.

See Also Window object

iGrafx API Object Hierarchy

{button View object,JI(`igrafxrf.HLP',`View_Object')}

Views Object

The Views object is a collection of View objects, which are views or windows within the iGrafx Professional application that are displaying data for a Document, a Component, or a Diagram object. View collections for each of these objects are separate; that is, the View collection for a Diagram object can only contain Diagram views. Component and Document View collections can contain any type of custom view.

Since custom views can be created for a diagram, component, or document (the Tabular view is an example of a custom view, and using the Extensions architecture, you can create additional custom views) a generic views collection is used to allow for these "custom" views.

A Views collection is associated with these objects:

- Document
- Diagram
- Component

If a particular view in the Views collection is iGrafx Professional's built-in view, the DiagramView, then the View's DiagramView property returns that DiagramView object. Otherwise, if the view is a custom view, the View's DiagramView property returns Nothing.

Properties, Methods, and Events

All of the properties, methods, and events for the Views object are listed in the following table. Click the name to view the documentation for any particular property, method, or event.

Properties	Methods	Events
Application	AddDiagramView	
Count	Item	
Parent		

Related Topics

Window object

View object

DiagramView object

iGrafx API Object Hierarchy

AddDiagramView Method

Syntax *Views*.**AddDiagramView**(*Diagram* As Diagram) As DiagramView

Description The AddDiagramView method adds a new DiagramView object to the Views collection. The *Diagram* argument specifies the name of the Diagram to be shown in the view.

If you access the View collection from a Document object, you must specify the *Diagram* argument when adding a new diagram view. If you access the View collection from a Diagram object, any value entered for the *Diagram* argument is ignored, because the value is provided implicitly. You cannot add a diagram view to the Views collection obtained from the Component object.

Example The following example adds two new diagram views to the Views collection. The first new view is added from the Diagram level, so no argument is required. The second new view is added from the Document level so the argument is required. Code is included that shows that the Views collection is common to both the Document and Diagram levels.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxDiagView As DiagramView
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
MsgBox "View the state of the diagram"
' Add a Diagram View to the Views collection, Diagram level access
Set igxDiagView = ActiveDiagram.Views.AddDiagramView()
' Verify that View collection is the same no matter which
' level it is accessed from
MsgBox "View the state of the diagram." & Chr(13) _
    & "The Views collection from the Document level contains " _
    & ActiveDocument.Views.Count & " views." & Chr(13) _
    & "The Views collection from the Diagram level contains " _
    & ActiveDiagram.Views.Count & " views."
' Add a Diagram View to the Views collection, Document level access
Set igxDiagView = ActiveDocument.Views.AddDiagramView(ActiveDiagram)
MsgBox "View the state of the diagram." & Chr(13) _
    & "The Views collection from the Document level contains " _
    & ActiveDocument.Views.Count & " views." & Chr(13) _
    & "The Views collection from the Diagram level contains " _
    & ActiveDiagram.Views.Count & " views."
```

See Also DiagramView object

{button Views object,JI(`igrafxrf.HLP',`Views_Object')}

Item Method

Syntax *Views*.**Item**

Description The Item method returns the View object at the specified *Index* from the Views collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type View. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example creates three views of the active diagram, and then uses the Item method to access each View object. Each view is activated and centered.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxDiagView As DiagramView
Dim igxView As View
Dim iCount As Integer
' Create the first shape on the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(2))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
MsgBox "View the state of the diagram"
' Add a Diagram View to the Views collection, Diagram level access
Set igxDiagView = ActiveDiagram.Views.AddDiagramView()
' Verify that View collection is the same no matter which
' level it is accessed from
MsgBox "View the state of the diagram." & Chr(13) _
    & "The Views collection from the Document level contains " _
    & ActiveDocument.Views.Count & " views." & Chr(13) _
    & "The Views collection from the Diagram level contains " _
    & ActiveDiagram.Views.Count & " views."
' Add a Diagram View to the Views collection, Document level access
Set igxDiagView = ActiveDocument.Views.AddDiagramView(ActiveDiagram)
MsgBox "View the state of the diagram." & Chr(13) _
    & "The Views collection from the Document level contains " _
    & ActiveDocument.Views.Count & " views." & Chr(13) _
    & "The Views collection from the Diagram level contains " _
    & ActiveDiagram.Views.Count & " views."
For iCount = 1 To ActiveDocument.Views.Count
    Set igxView = ActiveDiagram.Views.Item(iCount)
    Call igxView.Window.Activate
    Call igxView.Window.Center
    Application.RefreshUI
    MsgBox "View the result"
```

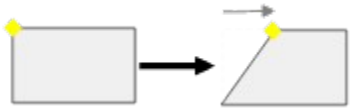
```
   Next iCount
```

{button Views object,JI(`igrafxrf.HLP',`Views_Object')}

Adjustment Object

The Adjustment object adds additional control points to a shape. A developer can use this object to add control points, which can then be monitored by code. For instance, when a user moves an adjustment control point, the shape can be changed in some way in response to the movement of the control points.

For example, the developer might add an adjustment to the top left corner of the shape that allows the user to bend the left side of a rectangle, as shown below.



An adjustment is simply an X, Y point in the coordinate space of the shape. The coordinate space of a shape is typically the bounding rectangle of the shape and coordinates go from 0 to 1. So the top left corner of a shape would be 0,0. The bottom right corner of a shape would be 1,1. The center point of a shape would be .5, .5. Negative values are left of the shape in the X direction, and above the shape in the Y direction. In the previous illustration, the adjustment point was placed at 0,0.

Note, however, that a developer can set up a custom coordinate space that is not the bounding rectangle of the shape. There are various reasons for using a custom coordinate space, but the most common is to make some aspect of programming with adjustments easier. For information about the coordinate space for shapes, see Shape Coordinate Space.

A developer can use adjustment points to change almost anything about a shape: it's color, it's text, etc. You are not limited to changing just the graphic, as shown in the following illustration.

Note that the shape's text does not have to appear within the boundaries of the shape (see the TextBlock object)
The Adjustment object is accessible only through the Shape object, by way of the Adjustments collection object (see iGrafx API Object Hierarchy).
Adding an adjustment point is done with the Adjustments.Add method. Setting the location of an adjustment point is done with the Adjustment.X and Adjustment.Y properties. Deleting an adjustment point is done with the Adjustment.Delete method.

The shape events directly related to adjustment points are the Shape_BeforeAdjustmentMove, Shape_AdjustmentMove, and Shape_AfterAdjustmentsMove events. A developer can write code for these events to make an adjustment point manipulate some aspect of the shape. Most commonly, adjustment points are used in conjunction with a shape's graphic. A programmer can get to a shape's graphic through the Shape.Graphic property.

Properties, Methods, and Events

All of the properties, methods, and events for the Adjustment object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
Parent		
X		
Y		

X Property

Syntax *Adjustment*.**X**

Data Type Double (read/write)

Description The X property specifies the current horizontal (X) position of the specified Adjustment point. The position is based on the relative coordinate space of the associated shape. Typically, X ranges from 0.0 to 1.0; however, a programmer can change the relative coordinate space, and thereby affect the value range for the property (see Shape Coordinate Space).

The following topics contain examples that show the X and Y properties of the Adjustment object:

- Adjustments.Add method—shows adding an adjustment point to a shape, and moving the adjustment point with the X and Y properties. Initial values for X and Y are specified using the arguments of the Add method.
- Shape_AdjustmentMove event—shows how the X and Y properties are read as part of a conditional statement.

{button Adjustment object,JI(`igrafxrf.HLP',`Adjustment_Object')}

Y Property

Syntax *Adjustment*.**Y**

Data Type Double (read/write)

Description The Y property specifies the current vertical position (Y) of the specified Adjustment point. The position is based on the relative coordinate space of the associated shape. Typically, Y ranges from 0.0 to 1.0; however, a programmer can change the relative coordinate space, and thereby affect the value range for Y (see Shape Coordinate Space).

The following topics contain examples that show the X and Y properties of the Adjustment object:

- Adjustments.Add method—shows adding an adjustment point to a shape, and moving the adjustment point with the X and Y properties. Initial values for X and Y are specified using the arguments of the Add method.
- Shape_AdjustmentMove event—shows how the X and Y properties are read as part of a conditional statement.

{button Adjustment object,JI(`igrafxrf.HLP',`Adjustment_Object')}

Adjustments Object

The Adjustments object is a collection of individual Adjustment objects. An Adjustments collection is associated with a shape (is accessible from the Shape object). Its purpose is to store and provide access to the individual Adjustment objects that have been created for a shape.

The Adjustments object provides the following functionality:

- The ability to access any Adjustment objects that have been created for a particular shape.
- The ability to determine how many Adjustment objects are in the collection.
- The ability to add a new Adjustment object to a shape.

Properties, Methods, and Events

All of the properties, methods, and events for the Adjustments object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent		

Related Topics

Adjustment Object

iGrafx API Object Hierarchy

Add Method

Syntax *Adjustments*.**Add**(*XOffset* As Double, *YOffset* As Double)

Description The Add method adds an adjustment point to the Adjustments collection for a particular shape. The *XOffset* and *YOffset* arguments specify the initial location of the adjustment point, relative to the Top, Left corner of the shape's bounding box.

Example The following example adds an adjustment point to the Adjustments collection of a new shape. The shape is then selected so you can see the adjustment point, and the point moved around the shape using the X and Y properties.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxAdjustments As Adjustments
' Set the igxDiagram variable to the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Create a new shape with its center at one inch and then
' set it to the igxShape variable
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, igxDiagram.DiagramType.ShapeLibrary.Item(1))
MsgBox "View the diagram"
' Set the igxAdjustments variable to Adjustments object
Set igxAdjustments = igxShape.Adjustments
' Add an adjustment point to the shape at the top and center
igxAdjustments.Add 0.5, 0.25
MsgBox "Adjustment point added"
' Select the shape
igxShape.DiagramObject.Selected = True
MsgBox "Shape selected so you can see the adjustment point"
' Move the adjustment point
For iCount = 1 To 4
   Select Case iCount
       Case 1:
           igxAdjustments.Item(1).X = 0.25
           igxAdjustments.Item(1).Y = 0.25
           MsgBox "Adjustment point moved"
       Case 2:
           igxAdjustments.Item(1).X = 0.25
           igxAdjustments.Item(1).Y = 0.75
           MsgBox "Adjustment point moved"
       Case 3:
           igxAdjustments.Item(1).X = 0.75
           igxAdjustments.Item(1).Y = 0.75
           MsgBox "Adjustment point moved"
       Case 4:
           igxAdjustments.Item(1).X = 0.75
           igxAdjustments.Item(1).Y = 0.25
           MsgBox "Adjustment point moved"
   End Select
Next iCount
```

{button Adjustments object,JI(`igrafxrf.HLP',`Adjustments_Object')}

Item Method

Syntax *Adjustments*.**Item**(*Index* As Integer) As Adjustment

Description The Item method returns the Adjustment object at the specified *Index* from the Adjustments collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Adjustment. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example adds a shape in the active diagram, and adds four adjustment points to the shape. Using the Item method and the Count property, each adjustment point is selected and then moved.

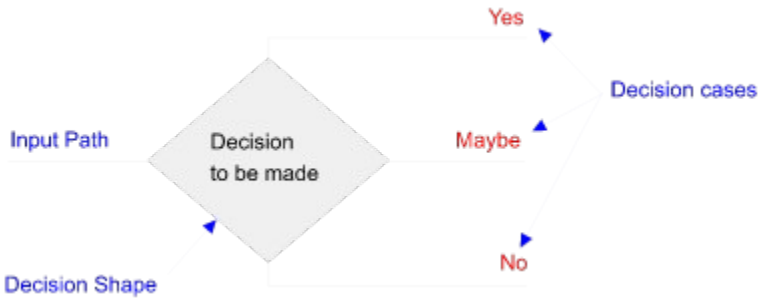
```
' Dimension the variables
Dim igxShape As Shape
Dim igxAdjustments As Adjustments
' Create a new shape with its center at one inch and then
' set it to the igxShape variable
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, igxDiagram.DiagramType.ShapeLibrary.Item(1))
MsgBox "View the diagram"
' Set the igxAdjustments variable to Adjustments object
Set igxAdjustments = igxShape.Adjustments
' Add four adjustment points to the shape
igxAdjustments.Add 0.25, 0.25
igxAdjustments.Add 0.75, 0.25
igxAdjustments.Add 0.25, 0.75
igxAdjustments.Add 0.75, 0.75
MsgBox "Adjustment points added"
' Select the shape
igxShape.DiagramObject.Selected = True
MsgBox "Shape selected so you can see the adjustment point"
' Use the Item method and Count property to select each
' adjustment point and move it
For iCount = 1 To igxAdjustments.Count
    Select Case iCount
        Case 1:
            igxAdjustments.Item(iCount).X = -0.25
            igxAdjustments.Item(iCount).Y = 0.25
            MsgBox "Adjustment point moved"
        Case 2:
            igxAdjustments.Item(iCount).X = 1.25
            igxAdjustments.Item(iCount).Y = 0.25
            MsgBox "Adjustment point moved"
        Case 3:
            igxAdjustments.Item(iCount).X = 0.25
            igxAdjustments.Item(iCount).Y = 1.25
            MsgBox "Adjustment point moved"
        Case 4:
            igxAdjustments.Item(iCount).X = 1.25
            igxAdjustments.Item(iCount).Y = 1.25
            MsgBox "Adjustment point moved"
    End Select
Next iCount
```

{button Adjustments object,JI(`igrafxrf.HLP',`Adjustments_Object')}

DecisionCase Object

The DecisionCase object defines a particular “decision branch” from a decision shape. The DecisionCase object is subordinate to the Shape object (through the DecisionCases collection object), and is only associated with shapes.

A decision shape is any shape that has two or more decision cases defined. You do this programatically by using the DecisionCases.Add method. The default decision shape in iGrafx Professional is a diamond. The following diagram helps illustrate the purpose of this object.



Each output from the decision shape is a decision case. In this example, Yes, No, and Maybe are the decision cases, and are associated with the Decision Shape (lets call it Shape1). The DecisionCases object is a collection of all the individual DecisionCase objects associated with a shape.
For information about decision shapes and their use within iGrafx Professional, refer to the iGrafx Professional User’s Guide. For an example of coding with the DecisionCase object, refer to the iGrafx System Developer’s Guide.

Properties, Methods, and Events

All of the properties, methods, and events for the DecisionCase object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
Name		
Parent		
Percent		

Related Topics

DecisionCases object
iGrafx API Object Hierarchy

Percent Property

Syntax *DecisionCase*.**Percent**

Data Type Double (read/write)

Description The Percent property specifies the output distribution (as a percentage) for a DecisionCase object. For example, a decision shape may have two decision cases as output: Yes and No. Use this property to set the percentage of the total output from the decision shape that gets routed to each decision case (for instance, 60% to Yes and 40% to No).

Example The following example uses the Percent property of a DecisionCase object to specify the percentage of the output that is routed to each of the two decision cases for the shape.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDecisionCases As DecisionCases
Dim igxDecisionCase As DecisionCase
' Create a new shape with its center at one inch and then
' set it to the igxShape variable
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, ActiveDiagram.DiagramType.ShapeLibrary.Item(1))
' Create two decision cases for the new shape
Set igxDecisionCases = igxShape.DecisionCases
' This decision case is named "Mostly" and its initial Percent value is 65%
igxDecisionCases.Add "Mostly"
' This decision case is named "Sometimes" and its initial Percent value is 35%
igxDecisionCases.Add "Sometimes"
' Set the decision case percentages
' Set "Mostly" to 65%
igxDecisionCases.Item(1).Percent = 65
' Set "Sometimes" to 35%
igxDecisionCases.Item(2).Percent = 35
' Display the initial values for each decision case
For iCount = 1 To igxDecisionCases.Count
    Set igxDecisionCase = igxDecisionCases.Item(iCount)
    MsgBox igxDecisionCase.Name & " is set at " _
        & igxDecisionCase.Percent & " percent"
Next iCount
' Set each decision case to 50%
For iCount = 1 To igxDecisionCases.Count
    Set igxDecisionCase = igxDecisionCases.Item(iCount)
    igxDecisionCase.Percent = 100 / igxDecisionCases.Count
    MsgBox igxDecisionCase.Name & " is set at " & _
        igxDecisionCase.Percent & " percent"
Next iCount
```

{button DecisionCase object,JI(`igrafxrf.HLP',`DecisionCase_Object')}

DecisionCases Object

The DecisionCases object is a collection of individual DecisionCase objects. A DecisionCases collection is an object property that is associated only with the Shape object. Its purpose is to store and provide access to the individual DecisionCase objects that have been created for a shape.

The DecisionCases object provides the following functionality:

- The ability to access any DecisionCase objects that have been created for a particular shape.
- The ability to determine how many DecisionCase objects are in the collection.
- The ability to add a new DecisionCase object to a shape.

Properties, Methods, and Events

All of the properties, methods, and events for the DecisionCases object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent		

Related Topics

DecisionCase object

iGrafx API Object Hierarchy

Add Method

Syntax *DecisionCases*.**Add**(*CaseName* As String) As DecisionCase

Description The Add method adds a DecisionCase object to the DecisionCases collection for a particular shape. The method returns a DecisionCase object.

Example The following example adds two DecisionCase objects to a shape, and then displays their names.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxDecisionCases As DecisionCases
Dim igxDecisionCase As DecisionCase
' Create a new shape with its center at one inch and then
' set it to the igxShape variable
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440, 1440, ActiveDiagram.DiagramType.ShapeLibrary.Item(1))
' Create two decision cases for the new shape
Set igxDecisionCases = igxShape.DecisionCases
' Add decision case named "Mostly"
Set igxDecisionCase = igxDecisionCases.Add("Mostly")
' Add decision case named "Sometimes"
Set igxDecisionCase = igxDecisionCases.Add("Sometimes")
' Display the names of all decision cases in the collection
For iCount = 1 To igxDecisionCases.Count
    Set igxDecisionCase = igxDecisionCases.Item(iCount)
    MsgBox igxDecisionCase.Name & " is Item " _
        & iCount & " in the collection"
Next iCount
```

{button DecisionCases object,JI(`igrafxrf.HLP',`DecisionCases_Object')}

Item Method

Syntax *DecisionCases*.**Item**(*Index* As Integer) As DecisionCase

Description The Item method returns the DecisionCase object at the specified *Index* from the DecisionCases collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type DecisionCase. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example Refer to the Example given for the Add method.

{button DecisionCases object,JI(`igrafxrf.HLP',`DecisionCases_Object')}

Change Object

The Change object is used with the ChangeBracket object to create user-defined Undo and Redo routines. The programmer must implement any custom Undo and Redo function (similar to the implimentation of the CommandHandler object). Once the Change object has been implemented, the StoreChange method of the ChangeBracket object is used to insert the Change object's methods into the Undo/Redo chain. Then, when an Undo or Redo is triggered within a ChangeBracket, the Change object's user-defined methods are called as part of the ChangeBracket's Undo or Redo operation. This allows the programmer to handle changes that may need to be rolled back (for example a database update) when the user has requested an Undo or Redo on an object with customized behavior.

Properties, Methods, and Events

All of the properties, methods, and events for the Change object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
	Redo	
	Undo	

Redo Method

Syntax *Change*.**Redo**

Description The Redo method is required in a Change Class—a class that Implements a Change object. If a user selects **Edit->Redo**, this method executes. The Redo is defined by the programmer to perform custom Redo operations during a ChangeBracket.

Example The following example sets up a Change Class that displays messages when Undo and Redo are performed. This is a simple example to show how a change class is implemented. The Undo and Redo methods can do much more, such as update databases or external programs based on the Undo and Redo activity in iGrafx Professional.

The first block of code is the Change Class. To make a new change class called "Class1", select the **Insert->New Class** menu item in the Visual Basic editor. Put this code inside the Class1 code window.

```
' Make this class a Change class
Implements Change
' Do this if the user selects Edit->Redo
Private Sub Change_Redo()
    MsgBox "The three shapes are about to be restored."
End Sub
' Do this if the user selects Edit->Undo
Private Sub Change_Undo()
    MsgBox "The three shapes are about to be removed."
End Sub
```

The remaining code block is an example routine that adds three shapes to the diagram, sets up a ChangeBracket, and associates our Class1 Change object with the ChangeBracket. After that, when the user tries to Undo or Redo, the custom Class1 Undo and Redo methods are executed. Put this block of code in a Diagram project, and run it.

```
Private Sub Main()
   ' Dimension thr variables
   Dim igxDiagram As Diagram
   Dim igxChangeBracket As ChangeBracket
   Dim Index As Integer
   ' Create a Change object derived from our
   ' custom "Class 1" Change Class
   Dim igxChange As New Class1
   ' Start a ChangeBracket
   Set igxChangeBracket = _
       ActiveDocument.OpenChangeBracket("MyChangeBracket")
   ' Create 3 new shapes
   MsgBox "Click OK to add 3 new shapes."
   For Index = 1 To 3
       ActiveDiagram.DiagramObjects.AddShape 1000 * Index, 1000 * Index
   Next Index
   ' Associate our custom "Class 1" igxChange object
   ' with this ChangeBracket
   igxChangeBracket.StoreChange igxChange
   igxChangeBracket.Close
   MsgBox "Change class ready. Try Undo and Redo."
End Sub
```

See Also

Undo method

ChangeBracket object

{button Change object,JI(`igrafxrf.HLP',`Change_Object')}

Undo Method

Syntax *Change*.**Undo**

Description The Undo method is required in a Change Class—a class that Implements a Change object. If a user selects **Edit->Undo**, this method executes. The Undo is defined by the programmer to perform custom Undo operations during a ChangeBracket.

Example Refer to the example for the Redo method for more information on how to implement a Change Class that handles ChangeBracket Undo and Redo routines.

See Also Redo method

ChangeBracket object

{button Change object,JI(`igrafxrf.HLP',`Change_Object')}

ChangeBracket Object

The ChangeBracket object is used to isolate a series of commands to speed processing and also to allow the program to create an entire block of commands that can be undone. Normally an undo will reverse only the last action. If you want to undo an entire series of commands, for example a series of formatting commands, you would have to undo them all one at a time. On the other hand, if you do all of the commands in a change bracket, all you need to do to undo all of the commands is select or specify undo once, because they are 'bracketed' together as one big undo. Processing is also enhanced because the commands in a change bracket are post processed unless otherwise specified.

Properties, Methods, and Events

All of the properties, methods, and events for the ChangeBracket object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Close	
Parent	StoreChange	
Repaint		
UpdateImmediately		

Close Method

Syntax *ChangeBracket*.**Close**

Description The Close method closes the specified ChangeBracket object. Closing a ChangeBracket ends the sequence of actions in the ChangeBracket. The ChangeBracket appears in the Edit menu as an Undo option after using the Close method.

Example The following example shows the Close method.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxChangeBracket As ChangeBracket
Dim index As Integer
' Set the igxDiagram variable to the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Create a ChangeBracket
Set igxChangeBracket = _
   Application.ActiveDocument.OpenChangeBracket("MyChangeBracket")
' Create some new shapes
For index = 1 To 5
    igxDiagram.DiagramObjects.AddShape _
    720 * index, 720 * index, igxDiagram.DiagramType.ShapeLibrary.Item(1)
Next index
igxChangeBracket.Close
' Now look at the Edit menu in iGrafx Professional
' Clicking the "Undo MyChangeBracket" command will
' undo all changes made to the active document during
' this Change Bracket
```

{button ChangeBracket object,JI(`igrafxrf.HLP',`ChangeBracket_Object')}

Repaint Property

Syntax *ChangeBracket*.**Repaint**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Repaint property specifies whether a diagram is repainted at the end of a ChangeBracket Undo or Redo. Set Repaint to True if an Undo or Redo could potentially leave unwanted graphical artifacts on the diagram. If set to True, and the user selects Edit->Undo or Redo for a ChangeBracket item, the actions are performed, and then the diagram is repainted. If set to False, Undo and Redo actions are performed, but any remaining diagram objects are not repainted afterward.

Example The following example shows the Repaint property.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxChangeBracket As ChangeBracket
Dim index As Integer
' Set the igxDiagram variable to the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Create a ChangeBracket
Set igxChangeBracket = _
Application.ActiveDocument.OpenChangeBracket("MyChangeBracket")
' Set the Repaint property to True
igxChangeBracket.Repaint = True
' Create some new shapes
For index = 1 To 5
    igxDiagram.DiagramObjects.AddShape _
    720 * index, 720 * index, igxDiagram.DiagramType.ShapeLibrary.Item(1)
Next index
igxChangeBracket.Close
' Now look at the Edit menu in iGrafx Professional
' Clicking the "Undo MyChangeBracket" command will
' undo all changes made to the active document during
' this Change Bracket
```

{button ChangeBracket object,JI(`igrafxrf.HLP',`ChangeBracket_Object')}

StoreChange Method

Syntax *ChangeBracket*.**StoreChange**(*pChange* As Change)

Description The StoreChange method associates the ChangeBracket object with a Change class object. A Change class has Undo and Redo methods that are executed when the user clicks the ChangeBracket object's Undo or Redo menu items in the application.

The *pChange* argument is a programmer-defined Change class object. A Change class must use the `Implement` keyword to "`Implement Change`" (see the example below), and it must have two methods: `Change_Redo`, and `Change_Undo`.

Example The following example sets up a Change Class that displays messages when Undo and Redo are performed. This is a simple example to show how a change class is implemented. The Undo and Redo methods can do much more, such as update databases or external programs based on the Undo and Redo activity in iGrafx Professional.

The first block of code is the Change Class. To make a new change class called "Class1", select the **Insert->New Class** menu item in the Visual Basic editor. Put this code inside the Class1 code window.

```
' Make this class a Change class
Implements Change
' Do this if the user selects Edit->Redo
Private Sub Change_Redo()
    MsgBox "The three shapes are about to be restored."
End Sub
' Do this if the user selects Edit->Undo
Private Sub Change_Undo()
    MsgBox "The three shapes are about to be removed."
End Sub
```

The remaining code block is an example routine that adds three shapes to the diagram, sets up a ChangeBracket, and associates the Class1 Change object with the ChangeBracket. After that, when the user tries to Undo, or Redo, our custom Class1 Undo and Redo methods are executed. Put this block of code in a Diagram code window, and run it.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxDiagram As Diagram
    Dim igxChangeBracket As ChangeBracket
    Dim Index As Integer
    ' Create a Change object derived from our
    ' custom "Class 1" Change Class
    Dim igxChange As New Class1
    ' Set the igxDiagram variable to the ActiveDiagram object
    Set igxDiagram = Application.ActiveDiagram
    ' Start a ChangeBracket
    Set igxChangeBracket = _
       ActiveDocument.OpenChangeBracket("MyChangeBracket")
    ' Create 3 new shapes
    MsgBox "Click OK to add 3 new shapes."
    For Index = 1 To 3
       igxDiagram.DiagramObjects.AddShape 1000 * Index, 1000 * Index
    Next Index
    ' Associate our custom "Class 1" igxChange object
```

```
    ' with this ChangeBracket
    igxChangeBracket.StoreChange igxChange
    igxChangeBracket.Close
    MsgBox "Change class ready. Try Undo and Redo."
End Sub
```

See Also Change object

{button ChangeBracket object,JI(`igrafxrf.HLP',`ChangeBracket_Object')}

UpdateImmediately Property

Syntax *ChangeBracket*.**UpdateImmediately**[= {**True | False**}]

Data Type Boolean (read/write)

Description The UpdateImmediately property causes the diagram to update immediately after a ChangeBracket Undo or Redo. The property allows the programmer to control when a diagram is updated. If set to True, diagram attributes such as line routes are recalculated immediately after an Undo or Redo is chosen by a user (this is the normal behavior). If set to False, a recalculation is not performed.

Example The following example demonstrates the UpdateImmediately property.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxChangeBracket As ChangeBracket
Dim index As Integer
' Set the igxDiagram variable to the ActiveDiagram object
Set igxDiagram = Application.ActiveDiagram
' Create a ChangeBracket
Set igxChangeBracket = _
Application.ActiveDocument.OpenChangeBracket("MyChangeBracket")
' Set the UpdateImmediately property to True
igxChangeBracket.Repaint = True
igxChangeBracket.UpdateImmediately = True
' Create some new shapes
For index = 1 To 5
    igxDiagram.DiagramObjects.AddShape _
       720 * index, 720 * index, igxDiagram.DiagramType.ShapeLibrary.Item(1)
Next index
igxChangeBracket.Close
' Now look at the Edit menu in iGrafx Professional
' Clicking the "Undo MyChangeBracket" command will
' undo all changes made to the active document during
' this Change Bracket
```

{button ChangeBracket object,JI(`igrafxrf.HLP',`ChangeBracket_Object')}

Component Object

A Component object is either a Scenario or a Report within a document. A Document object can contain multiple Scenarios and Reports, which are stored in the Components collection object. The Component object is subordinate to the Document object, and is accessed through the Components collection object. Components are only associated with Document objects.

At the component-level, the developer cannot change the contents of Scenarios or Reports, but superficial properties such as Name and Views can be manipulated. Components cannot be added to a document through VBA automation, but they can be deleted, using the DeleteComponent method. Component objects have a number of VBA events that occur when a user works with components. The developer can monitor these events to respond when Components are activated, opened, closed, renamed, etc.

Components have items that allow the developer to extend the object model: the *UserEvent* event for designing custom events; and the *AsType* property, for designing components with custom properties and methods. These work similar to other objects in the iGrafx Professional system that use UserEvent and AsType. See the UserEvent and AsType topics for more information.

Properties, Methods, and Events

All of the properties, methods, and events for the Component object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	DeleteComponent	Activate
AsType	FireUserEvent	Close
ClassID		Deactivate
Name		Delete
PageLayout		GetInterface
Parent		New
ProgID		Open
Views		PageLayoutChange
		Print
		Rename
		Save
		UserEvent

Related Topics

Components object

iGrafx API Object Hierarchy

Activate Event

Syntax **Private Sub** *Component*_**Activate**()

Description The Activate event occurs when the specified component is activated; that is, the component obtains the focus. Custom code can be written within this event procedure to perform any desired actions. This event could be useful for customizing the user interface for a particular component on activation.

Example The following example shows where to place Component event code. This example uses the Activate method to display a message when the user views the Report.

First set up a component:

1. On the Model Toolbar, click Components
2. In the Components dialog, click New->Report

Now go to the Visual Basic Editor Project Explorer:

1. Right-Click Report1
2. Select View Code
3. Add the following event code:

```
Private Sub Component_Activate()
    MsgBox "Thank you for viewing Report1."
End Sub
```

Now when you click on the Report1 window, the event fires.

See Also Deactivate event

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

AsType Property

Syntax *Component*.**AsType**(*TypeName* As String) As Object

Data Type Object (read-only)

Description The AsType property allows you to add your own properties and methods to a Component object, extending the object model. The properties and methods can be organized into one or more component types, using unique type names.

The *TypeName* argument is a string that names the custom type. It can be any string you choose, but it must be unique within the environment. In an integrated environment, other programmers may be accessing the Component, and using it's AsType property. To prevent conflicting type names, it is suggested that you use your company or department name, followed by a descriptive type name (for example, "MyCompanyFactory")

Use the following basic steps to implement a custom property or method for the Document object.

1. Use Component.AsType ("my type name").MyMethod in your code.
2. Create a new Class, and design properties and methods in the class.
3. Set up the GetInterface event to check the TypeName string passed to it. If it matches your type name, set the Interface parameter equal to your new class.

When you use Component.AsType(*TypeName*) in your code, you gain access to the properties and methods that you have defined in the new Class. The Component.AsType property automatically fires an event called GetInterface. The GetInterface event can have one or more AsType's defined, each one distinguished by a unique type name. Based on the type name, the GetInterface event redirects execution to your new Class by setting the Interface parameter. If the Interface parameter is set to your new Class, the Class properties and methods become exposed to the Component object.

Example The following example shows the basic design for implementing the Component.AsType property. Put each block of code into the code window indicated. Run the subroutine in "ThisApplication" to see the AsType property work.

Document – Class1 (Code)

```
Public Property Get AircraftType()
    AircraftType = "Boeing 747"
End Property
```

Document – Report 1 (Code)

```
Private Sub Main()
    MsgBox "The aircraft is a " & Report1.AsType("Airplane").AircraftType
End Sub

Private Sub Component_GetInterface(ByVal TypeName As String, _
Interface As Object)
    'Check the TypeName
    If TypeName = "Airplane" Then
        ' If the Interface Is Nothing, its available to Set
        If Interface Is Nothing Then
            ' Return our Class1 as the Interface
            Set Interface = New Class1
        End If
    End If
```

```
End Sub
```

See Also GetInterface event

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

ClassID Property

Syntax *Component*.**ClassID**

Data Type String (read-only)

Description The ClassID property returns the class ID of the component as resident in the Windows environment. Every class in the system has a unique Class ID.

Example The following example displays the Class ID of a Report. The preceding instructions add a Report component to the project.

First set up a component:

1. On the Model Toolbar, click Components
2. In the Components dialog, click New->Report
3. Now add the following code to the "ThisDocument" code windows

```
Private Sub Main()
    Dim igxReport1 As Component
    Set igxReport1 = ActiveDocument.Components.Item(1)
    MsgBox "Report1 Class ID is: " & igxReport1.ClassID
End Sub
```

See Also ProgID property

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

Close Event

Syntax *Component*.**Close**()

Description The Close event occurs when the specified component is closed, either as a result of code you have written, or a user action. Custom code can be written within this event procedure to perform any desired actions. This event could be useful for clean-up operations for a particular component.

Example The following example uses the Open and Close events. The Close event displays the total number of seconds the component was opened. It calculates the duration by comparing the system time when opened, with the system time when closed. This code goes into an existing component code window, and the document must be saved to disk. Opening and closing the document from disk fires these events.

```
' Dimension a module variable to store time in seconds
Private TimeOpened As Long

' The Open event stores the current system time
Private Sub Component_Open()
   TimeOpened = Timer
End Sub

' The Close event displays the duration opened
Private Sub Component_Close()
   ' Dimension the variables
   Dim Duration As Long
   ' Calculate duration open
   Duration = Timer - TimeOpened
   ' Display the result, and ask the user to confirm closing
   MsgBox "The component was opened for " & Int(Duration) _
       & " seconds."
End Sub
```

See Also Open event

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

Deactivate Event

Syntax *Component*.**Deactivate**()

Description The Deactivate event occurs when the specified component is deactivated; that is, the component loses the focus. A deactivation event can occur as a result of code you have written, or from user actions. Custom code can be written within this event procedure to perform any desired actions. This event could be useful for clean-up operations for a particular component.

Example The following example shows where to place Component event code. The example uses the Deactivate method to display a message when the user deactivates the Report.

First set up a component:

1. On the Model Toolbar, click Components
2. In the Components dialog, click New->Report

Now go to the Visual Basic Editor Project Explorer:

1. Right-Click Report1
2. Select View Code
3. Add the following event code:

```
Private Sub Component_Deactivate()
    MsgBox "Report1 will now be behind another window."
End Sub
```

Now when you deactivate the Report1 window, the event fires.

See Also Activate event

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

Delete Event

Syntax *Component*.**Delete**()

Description The Delete event occurs when a component is deleted. Components can be deleted either programmatically or by user actions. Custom code can be written within this event procedure to perform any desired actions.

Example The following example shows where to place Component event code. The example uses the Delete method to store the time that the Report was deleted.

First set up a component:

1. On the Model Toolbar, click Components
2. In the Components dialog, click New->Report

Now go to the Visual Basic Editor Project Explorer:

1. Right-Click Report1
2. Select View Code
3. Add the following event code:

```
Public WhenDeleted As Date
Private Sub Component_Delete()
    MsgBox "Report1 deleted " & Now
   WhenDeleted = Now
End Sub
```

Now when you delete the Report1 window, the event fires.

See Also DeleteComponent method

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

DeleteComponent Method

Syntax *Component*.**DeleteComponent**

Description The DeleteComponent method deletes the specified Component object.

Example The following example shows where to place Component event code. This example uses the Delete method to store the time that the Report was deleted.

First set up a component:

1. On the Model Toolbar, click Components
2. In the Components dialog, click New->Report

Now add this code to the "ThisDocument" code windows and run it.

```
' Dimension the variables
Dim igxReport1 As Component
' Get the Report1 component object
Set igxReport1 = ActiveDocument.Components.Item(1)
' Delete the component
MsgBox "Click OK to delete Report1."
igxReport1.DeleteComponent
MsgBox "Click Ok to continue."
```

See Also Delete event

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

FireUserEvent Method

Syntax *Component*.**FireUserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The FireUserEvent method fires the "UserEvent" for the specified component. You can use this functionality to send messages to any component that is listening to events.

You must specify an *EventIdentifier* argument (a string) to use for your event. You might choose to use something like your company name followed by the event name. You should choose a name that won't conflict with names picked by other developers.

You can pass one parameter to the event (the *Parameter* argument). This parameter is a Variant, so one logical choice is to pass a Class.

Then, you can write code in a UserEvent handler to perform some actions when your event fires. This code should be of the form:

```
If EventIdentifier = "<<Your identifier string>>" Then
   << Write your code here >>
End If
```

Example The following example defines a new user event called "ShowUsers". The Parameter that gets passed is a class, which has one property called Count. The event handler displays the passed parameter's Count property.

The following code implements a simple class with one property. Create a new class below a diagram project called Class1 and copy this code into it.

```
' Class1
' It contains one property, read only
Public Property Get Count() As Long
    Count = 25
End Property
```

The following is the main program. Copy this, and the UserEvent subroutine, into the diagram project code window

```
' Run this subroutine to test the event
Public Sub Main()
    ' Create a new Class1 object
    Dim MyClass1 As New Class1
    ' Fire the UserEvent
    Report1.FireUserEvent "ShowUsers", MyClass1
End Sub

' This event handler runs every time the FireUserEvent method
' is used on the component
Private Sub Component_UserEvent(ByVal EventIdentifier As String, ByVal
Parameter As Variant)
    ' Check if the Identifier string is the one we want
    If EventIdentifier = "ShowUsers" Then
        ' Redirect to Class1
        MsgBox "The number of users is " & Parameter.Count
    End If
End Sub
```

See Also UserEvent event

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

GetInterface Event

Syntax **Private Sub** *Component*_**GetInterface**(ByVal *TypeName* As String, *Interface* As Object)

Description The GetInterface event occurs when the Component.AsType property is used. The AsType property allows you to add your own properties and methods to a Component object, extending the object model. The properties and methods can be organized into one or more component types, using unique type names.

The *TypeName* argument is a string that distinguishes the custom type. It can be any string the programmer chooses, but it must be unique within the environment. In an integrated environment, other programmers may be accessing the component, and using it's AsType property. To prevent conflicting type names, it is suggested that you use your company or department name, followed by a descriptive type name (for example, "MyCompanyFactory").

Use the following basic steps to implement a custom property or method for the Component object.

1. Use Component.AsType ("my type name").MyMethod in your code.
2. Create a new Class, and design properties and methods in the class.
3. Set up the GetInterface event to check the TypeName string passed to it. If it matches your type name, set the Interface parameter equal to your new class.

When you use Component.AsType(*TypeName*) in your code, you gain access to the properties and methods that you have defined in the new Class. The Component.AsType property automatically fires an event called GetInterface. The GetInterface event can have one or more AsType's defined, each one distinguished by a unique type name. Based on the type name, the GetInterface event redirects execution to your new Class by setting the Interface parameter. If the Interface parameter is set to your new Class, the Class properties and methods become exposed to the Component object.

Notes When you extend an iGrafx Professional object using the GetInterface event, you need to keep in mind that other developers may be using this event also. To be a good citizen, you should do the following:

- Be sure to pick a name that is likely to be unique for your AsType name. In the example above, "MyType" is too generic and it is possible that another developer could use the same name. Instead, follow the convention of using your name or your company name, a period, and a description of the type. For example, if you were writing a type that extended Application to add additional internet capabilities, and your company name was "Micrografx", you could name your AsType name "Micrografx.InternetExtension".
- When you write code in the GetInterface event, keep it simple. You should not do any time consuming operation in the GetInterface event such as querying a database or displaying a dialog box.
- When you write code in the GetInterface event, be aware of the current state of the Interface parameter. In the example above, this is illustrated by the code fragment "Interface Is Nothing". If this code fragment evaluates to true, then it is safe to Set the interface to your class. If this code fragment evaluates to false then someone else has already responded to the event and set the interface to their class. If this condition arises, you should try changing your AsType name.

Example Refer to the Example for the AsType property for more information on using AsType with the GetInterface event.

See Also AsType property

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

New Event

Syntax *Component*.**New**()

Description The New event occurs when a new component is created. Custom code can be written within this event procedure to perform any desired actions. The New event must go into a Template. Create a template that has one open component as part of the Template project. Put the New event into a Component code window. Then, when a new document is created using that Template, the New event fires.

Example The following example stores the date and time that a new component was created. This event code must go into a Component code window that is part of a Template project.

```
Private Sub Component_New()
    Public WhenComponentCreated As Date
    WhenComponentCreated = Now
    MsgBox "This component was created " & WhenComponentCreated
End Sub
```

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

Open Event

Syntax *Component*.**Open**()

Description The Open event occurs when a document is opened from disk. This opens the component as well, firing the event. The Open event must be in a Component code window, and the Component must be in a document that has been saved to disk.

Example The following example uses the Open and Close events. The Close event displays the total number of seconds the component was opened. It calculates the duration by comparing the system time when opened, with the system time when closed. This code goes into an existing component code window, and the document must be saved to disk. Opening and closing the document from disk fires these events.

```
' Dimension a module variable to store time in seconds
Private TimeOpened As Long

' The Open event stores the current system time
Private Sub Component_Open()
    TimeOpened = Timer
End Sub

' The Close event displays the duration opened
Private Sub Component_Close()
    ' Dimension variable
    Dim Duration As Long
    ' Calculate duration open
    Duration = Timer - TimeOpened
    ' Display the result, and ask the user to confirm closing
    MsgBox "The component was opened for " & Int(Duration) _
        & " seconds."
End Sub
```

See Also Close event

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

PageLayout Property

Syntax *Component*.**PageLayout**

Data Type PageLayout object (read-only, See Object Properties)

Description The PageLayout property returns a PageLayout object for the specified Component object. This property allows you to set up the page layout for the Component object, such as a Report or Scenario. The page layout affects the appearance of the Component when it is printed on a printer.

Example The following example uses a Report's PageLayout object to change the print orientation from Portrait to Landscape, and displays the result in the Print Preview.

```
' Add a report called Report1 and put this
' code into it's code window
' Dimension the variables
Dim igxPageLayout As PageLayout
' Get Report1's PageLayout object
Set igxPageLayout = Report1.PageLayout
' Set the print orientation to Landscape
igxPageLayout.Orientation = ixPageLandscape
' Activate Report1's window
ActiveDiagram.ActivateDiagram
Report1.Views.Item(2).Window.Activate
MsgBox "Click OK to view PrintPreview."
' Display the Print Preview
ExecuteCommand (ixFilePrintPreview)
MsgBox "Report1 orientation changed to LandScape."
```

See Also PageLayout object

iGrafx API Object Hierarchy

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

PageLayoutChange Event

Syntax *Component*_**PageLayoutChange**(*ClassID* As String)

Description The PageLayoutChange event occurs when a PageLayout property of the specified Component object has been modified. This is useful when the user wants to be informed of user changes to a component's page layout.

The *ClassID* parameter contains the ClassID of the View object whose page layout changed.

Example The following example makes a change to Report1's PageLayout, which fires the event. The event displays the ClassID of the view that contains Report1.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxPageLayout As PageLayout
   ' Get Report1's PageLayout object
   Set igxPageLayout = Report1.PageLayout
   ' Set the print orientation to Landscape
   igxPageLayout.Orientation = ixPagePortrait
End Sub

Private Sub Component_PageLayoutChange(ByVal ClassID As String)
   ' Display the view ClassID
   MsgBox "Report1's page layout has changed." & Chr(13) & _
       "The View ClassID is:" & ClassID
End Sub
```

See Also PageLayout property

View object

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

Print Event

Syntax *Component*.**Print**()

Description The Print event occurs when the specified component is printed (the “Print” command has been issued for that component). The event occurs as a result of calling a “Print” method from the API, or from a user action. Custom code can be written within this event procedure to perform any desired actions.

Example The following example prints the active Report. The Report is printed because it is on top when the application Print command is executed. This fires the event, which sets the Report's orientation to Portrait for printing

```
' This sample code goes into Report1's code window
Private Sub Main()
   ' Activate Report1's window
   Report1.Views.Item(2).Window.Activate
   ' Execute the application Print command
   ExecuteCommand (ixFilePrint)
End Sub

Private Sub Component_Print()
   ' Set Report1's orientation to Portrait
   Report1.PageLayout.Orientation = ixPagePortrait
   MsgBox "Report1's orientation has been set to Portrait for printing."
End Sub
```

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

ProgID Property

Syntax *Component*.**ProgID**

Data Type String (read-only)

Description The ProgID property returns a string that identifies which program the component belongs to, and the type of component. For instance, if the component is a Report, the property returns "iGrafx.Report". If it's a Scenario, the property returns "iGrafx.Scenario."

Example The following example lists each component in the document and reports it's type.

```
' Add as many reports and scenarios to the document
' as you want. Then try this example code.
' Dimension the variables
Dim igxComponents As Components
Dim sString As String
' Get the document's Components collection
Set igxComponents = ActiveDocument.Components
' Determine each Component object's type and store it in a string
For Index = 1 To igxComponents.Count
   If igxComponents.Item(Index).ProgID = "iGrafx.Report" Then
       sString = sString & "Component No." & Str(Index) & _
           " is a Report." & Chr(13)
   Else
       sString = sString & "Component No." & Str(Index) & _
           " is a Scenario." & Chr(13)
   End If
Next Index
' Display the result
MsgBox sString
```

See Also ClassID property

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

Rename Event

Syntax *Component*.**Rename**(ByVal *OldName* As String)

Description The Rename event occurs when the specified component is renamed, either as a result of code you have written or by user action. Custom code can be written within this event procedure to perform any desired actions.

The *OldName* parameter is a string that contains the name of the component before it was renamed.

Example The following example implements the Rename event. The Main( ) subroutine renames the Report, which fires the event. The event displays the Report that was renamed, and what the name was changed to.

```
Private Sub Main()
   ' Rename Report1
   Report1.Name = "Yesterday's Report"
End Sub

Private Sub Component_Rename(ByVal OldName As String)
   ' Dimension the variables
   Dim qt As String
   qt = Chr(34) 'quote character
   MsgBox "Component " & qt & OldName & qt & _
       " has been renamed to " & qt & Report1.Name & qt
End Sub
```

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

Save Event

Syntax *Component*.**Save**()

Description The Save event occurs when the specified component is saved with it's document, either as a result of code you have written or by user action. A Component is saved when it's parent document is saved. Custom code can be written within this event procedure to perform any desired actions.

Example The following example adds a Property to the Report object called LastTimeSaved. Whenever the parent document and component are saved to disk, the Save event stores the date and time of the save. The Main( ) subroutine retrieves the value and displays the LastTimeSaved

```
' Dimension a module variable to store the last time saved
Private SaveTime As String

' Add a Property to the Report object called LastTimeSaved
Public Property Get LastTimeSaved() As String
   LastTimeSaved = SaveTime
End Property

Private Sub Main()
   ' Check Report1's LastTimeSaved property
   If Report1.LastTimeSaved = "" Then
       ' If the property is Nothing, it hasn't been saved
       MsgBox "The component has not yet been saved."
   Else
       ' Otherwise display the last time saved
       MsgBox "The component was last saved " & _
           Report1.LastTimeSaved
   End If
End Sub

' When the component is saved, store the date and time
Private Sub Component_Save()
   SaveTime = Now
End Sub
```

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

UserEvent Event

Syntax **Private Sub** *Component*_**UserEvent**(*EventIdentifier* As String, *Parameter* As Variant)

Description The UserEvent event provides a means of implementing your own custom events. Your custom events can then be triggered with the FireUserEvent method, which fires the specified "UserEvent" on the document. You can use this functionality to send messages to any objects listening to document-level events.

You must pick an event identifier string to use for your event. You might choose to use something like your company name followed by the event name. You should choose a name that won't conflict with names picked by other developers.

You can pass one parameter to the event. This parameter is a Variant, so one logical choice is to pass a class.

You then write code in a UserEvent handler to perform some actions when your event fires. This code should be of the form:

```
If EventIdentifier = "<<Your identifier string>>" Then
   << Write your code here >>
End If
```

Example The following example defines a new user event called "ShowUsers". The *Parameter* that gets passed is a class, which as one property called Count. The event handler displays the passed parameter's Count property.

The following code creates a simple class with one property. Create a new class below a diagram project called Class1 and copy this code into it.

```
' Class1
' It contains one property, read only
Public Property Get Count() As Long
   Count = 25
End Property
```

The following code is the main program. Copy this, and the UserEvent subroutine, into the diagram project code window

```
' Run this subroutine to test the event
Public Sub Main()
   ' Fire the UserEvent
   Diagram.FireUserEvent "ShowUsers", New Class1
End Sub

' This event handler runs every time any FireUserEvent method
' is used in the system
Private Sub Diagram_UserEvent(ByVal EventIdentifier As String, ByVal Parameter
As Variant)
   ' Check if the Identifier string is the one we want
   If EventIdentifier = "ShowUsers" Then
       ' Redirect to Class1
       MsgBox "The number of users is " & Parameter.Count
   End If
End Sub
```

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

Views Property

Syntax *Component*.**Views**

Data Type Views object (read-only, See Object Properties)

Description The Views property returns a Views collection for the specified Component object. The property allows you to establish and manipulate views onto Component objects such as Reports and Scenarios.

Example The following example accesses Report1's View object, and uses it to activate Report1's window. Now when PrintPreview is executed, Report1 shows up in the Print Preview.

```
' Add a report called Report1 and put this
' code into it's code window
'
' Activate Report1's window
ActiveDiagram.ActivateDiagram
Report1.Views.Item(2).Window.Activate
MsgBox "Click OK to view PrintPreview."
' Display the Print Preview
ExecuteCommand (ixFilePrintPreview)
MsgBox "Report1 orientation changed to LandScape."
```

See Also Views object

iGrafx API Object Hierarchy

{button Component object,JI(`igrafxrf.HLP',`Component_Object')}

Components Object

The Components object is a collection of individual Component objects. A Components collection is only associated with and accessible from the Document object. Its purpose is to store and provide access to the individual Reports and Scenarios that have been created for a document.

The Components object provides the following functionality:

- The ability to access any Component objects that exist for a particular Document object.
- The ability to determine how many Component objects are in the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the Components object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
NextSuggestedName		
Parent		

Related Topics

Component object

Document object

iGrafx API Object Hierarchy

Item Method

Syntax *Components*.**Item**(*Index* As Integer) As Component

Description The Item method returns the Component object at the specified *Index* from the Components collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Component. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example lists each component in the document and displays it's type.

```
' Add as many reports and scenarios to the document
' as you want. Then try this example code.
'
' Dimension the variables
Dim igxComponents As Components
Dim sString As String
' Get the document's Components collection
Set igxComponents = ActiveDocument.Components
' Determine each Component object's type and store it in a string
For Index = 1 To igxComponents.Count
   If igxComponents.Item(Index).ProgID = "iGrafx.Report" Then
       sString = sString & "Component No." & Str(Index) & _
           " is a Report." & Chr(13)
   Else
       sString = sString & "Component No." & Str(Index) & _
           " is a Scenario." & Chr(13)
   End If
Next Index
' Display the result
MsgBox sString
```

{button Components object,JI(`igrafxrf.HLP',`Components_Object')}

NextSuggestedName Property

Syntax *Components*.**NextSuggestedName**(*ComponentName* As String, *ClassID* As String)

Data Type String (read-only)

Description The NextSuggestedName property returns a Component name as a string. It builds a new name by combining an alphabetical prefix with a numeric suffix. The result is a generated name such as "ProductionReport3" or MyScenario5".

The *ComponentName* argument is the prefix to use when generating the name. Use any string you want for the prefix. The NextSuggestedName property uses this string as the prefix when generating the new name.

The *ClassID* argument is the ClassID string of a Report or a Scenario, which determines what sequence to look for. It is used to determine the next numeric suffix in the sequence. For instance, supply this argument with the ClassID of an existing Report, and if Report1 and Report2 already exist, the next generated name is Report3.

Example The following example generates the next suggested name for a report component. The NextSuggestedName property is supplied with the prefix string "Report", and a sequence to continue—Report1.ClassID.

```
' Dimension the variables
Dim igxComponents As Components
Dim ReportName As String
' Get the document's Components collection object
Set igxComponents = ActiveDocument.Components
' Store the generated name            Use this prefix  Use this sequence
ReportName = igxComponents.NextSuggestedName("Report", Report1.ClassID)
' Display the result
MsgBox "The next suggested Report name is " & ReportName
```

{button Components object,JI(`igrafxrf.HLP',`Components_Object')}

ComponentRange Object

The ComponentRange object is a collection of Component objects. Component objects are the Reports and Scenarios in a project. Use ComponentRange collections to group Component objects in some useful manner. For instance, you may want to collect all Reports into one ComponentRange, and all Scenarios into another ComponentRange.

Properties, Methods, and Events

All of the properties, methods, and events for the ComponentRange object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	AddRange	
Parent	Item	
	Remove	
	RemoveAll	
	RemoveRange	

Add Method

Syntax *ComponentRange*.**Add**(*Component* As Component)

Description The Add method adds a Component object to the specified ComponentRange object. The *Component* argument specifies the Component object to add.

Example The following example creates a ComponentRange and adds two Component objects to it. It then displays the contents of the ComponentRange.

```
' Add two Reports and two Scenarios to the document
' before trying this example
'
' Dimension the variables
Dim igxComponents As Components
Dim Report1 As Component
Dim Report2 As Component
Dim Scenario1 As Component
Dim Scenario2 As Component
Dim igxCompRange1 As ComponentRange
Dim igxCompRange2 As ComponentRange
' Get the Document's Components object
Set igxComponents = ActiveDocument.Components
Set igxCompRange1 = ActiveDocument.MakeComponentRange
Set igxCompRange2 = ActiveDocument.MakeComponentRange
' Add components to the range
igxCompRange1.Add igxComponents.Item(1)
igxCompRange1.Add igxComponents.Item(2)
' Collect the names of the components contained in the range
For Index = 1 To igxCompRange1.Count
    sString = sString & igxCompRange1.Item(Index).Name & Chr(13)
Next Index
' Display the result
MsgBox "ComponentRange1 contains these components:" _
    & Chr(13) & Chr(13) & sString
```

See Also AddRange method

{button ComponentRange object,JI(`igrafxrf.HLP',`ComponentRange_Object')}

AddRange Method

Syntax *ComponentRange*.**AddRange**(*Range* As ComponentRange)

Description The AddRange method adds the contents of one ComponentRange to the contents of another ComponentRange. The *Range* argument specifies which ComponentRange object's contents are added to the specified ComponentRange object. The AddRange method "copies" content references, it does not "move" them, so the contents of the ComponentRange given as the argument are not altered.

Error Supplying the *Range* argument with an invalid ComponentRange produces an error. Use error trapping if your code could potentially supply a *Range* argument that is invalid.

Example The following example adds one ComponentRange to another. The contents of each ComponentRange is displayed before and after adding.

```
Private Sub Main()
    ' Add two Reports and two Scenarios to the document
    ' before trying this example
    '
    ' Dimension the variables
    Dim igxComponents As Components
    Dim Report1 As Component
    Dim Report2 As Component
    Dim Scenario1 As Component
    Dim Scenario2 As Component
    Dim igxCompRange1 As ComponentRange
    Dim igxCompRange2 As ComponentRange
    ' Get the Document's Components object
    Set igxComponents = ActiveDocument.Components
    Set igxCompRange1 = ActiveDocument.MakeComponentRange
    Set igxCompRange2 = ActiveDocument.MakeComponentRange
    ' Add components to igxCompRange1
    igxCompRange1.Add igxComponents.Item(1)
    igxCompRange1.Add igxComponents.Item(2)
    ' Add components to igxCompRange2
    igxCompRange2.Add igxComponents.Item(3)
    igxCompRange2.Add igxComponents.Item(4)
    ' Display the intermediate result
    MsgBox "ComponentRange1 contains these components:" & Chr(13) _
        & NamesIn(igxCompRange1) & Chr(13) _
        & "ComponentRange2 containes these components:" & Chr(13) _
        & NamesIn(igxCompRange2)
    ' Add the second range to the first range
    MsgBox "Click OK to add the second range to the first."
    igxCompRange1.AddRange igxCompRange2
    ' Display the result
    MsgBox "ComponentRange1 contains these components:" & Chr(13) _
        & NamesIn(igxCompRange1) & Chr(13) _
        & "ComponentRange2 containes these components:" & Chr(13) _
        & NamesIn(igxCompRange2)
End Sub

' This function returns the names of all the components in a range
' separated with carriage returns.  This function is used many times
' in the Message Boxes in the Main() subroutine.
```

```
Private Function NamesIn(Range As ComponentRange) As String
    Dim Index As Integer
    Dim sString As String
    For Index = 1 To Range.Count
        sString = sString & Range.Item(Index).Name & Chr(13)
    Next Index
    NamesIn = Chr(13) & sString & Chr(13)
End Function
```

See Also Add method

{button ComponentRange object,JI(`igrafxrf.HLP',`ComponentRange_Object')}

Item Method

Syntax *ComponentRange*.**Item**(*Index* As Integer) As Component

Description The Item method returns the Component object at the specified *Index* from the ComponentRange collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Component. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example creates a ComponentRange and adds two Component objects to it by using the Item method to specify the desired component. It then displays the contents of the ComponentRange collection.

```
' Add two Reports and two Scenarios to the document
' before trying this example
'
' Dimension the variables
Dim igxComponents As Components
Dim Report1 As Component
Dim Report2 As Component
Dim Scenario1 As Component
Dim Scenario2 As Component
Dim igxCompRange1 As ComponentRange
Dim igxCompRange2 As ComponentRange
' Get the Document's Components object
Set igxComponents = ActiveDocument.Components
Set igxCompRange1 = ActiveDocument.MakeComponentRange
Set igxCompRange2 = ActiveDocument.MakeComponentRange
' Add components to the range
igxCompRange1.Add igxComponents.Item(1)
igxCompRange1.Add igxComponents.Item(2)
' Collect the names of the components contained in the range
For Index = 1 To igxCompRange1.Count
    sString = sString & igxCompRange1.Item(Index).Name & Chr(13)
Next Index
' Display the result
MsgBox "ComponentRange1 contains these components:" _
    & Chr(13) & Chr(13) & sString
```

{button ComponentRange object,JI(`igrafxrf.HLP',`ComponentRange_Object')}

Remove Method

Syntax *ComponentRange*.**Remove**(*Component* As Component)

Description The Remove method removes a Component object from the ComponentRange. The *Component* argument determines which Component object to remove.

Error Specifying an invalid Component object produces an error. Use error trapping if your code could potentially supply the Remove method with an invalid Component object.

Example The following example adds four Component objects to a ComponentRange. It then displays the contents of the range. Then one of the Components is removed using the Remove method, and the contents are again displayed.

```
Private Sub Main()
    ' Add two Reports and two Scenarios to the document
    ' before trying this example
    '
    ' Dimension the variables
    Dim igxComponents As Components
    Dim Report1 As Component
    Dim Report2 As Component
    Dim Scenario1 As Component
    Dim Scenario2 As Component
    Dim igxCompRange1 As ComponentRange
    ' Get the Document's Components object
    Set igxComponents = ActiveDocument.Components
    Set igxCompRange1 = ActiveDocument.MakeComponentRange
    ' Add components to igxCompRange1
    igxCompRange1.Add igxComponents.Item(1)
    igxCompRange1.Add igxComponents.Item(2)
    igxCompRange1.Add igxComponents.Item(3)
    igxCompRange1.Add igxComponents.Item(4)
    ' Display the intermediate result
    MsgBox "ComponentRange1 contains these components:" & Chr(13) _
        & NamesIn(igxCompRange1)
    ' Remove a component from the range
    MsgBox "Click OK to remove the last component."
    igxCompRange1.Remove igxComponents.Item(4)
    ' Display the result
    MsgBox "ComponentRange1 contains these components:" & Chr(13) _
        & NamesIn(igxCompRange1)
End Sub

' This function returns the names of all the components in a range
' separated with carriage returns
Private Function NamesIn(Range As ComponentRange) As String
    Dim Index As Integer
    Dim sString As String
    For Index = 1 To Range.Count
        sString = sString & Range.Item(Index).Name & Chr(13)
    Next Index
    NamesIn = Chr(13) & sString & Chr(13)
End Function
```

See Also RemoveAll method

{button ComponentRange object,JI(`igrafxrf.HLP',`ComponentRange_Object')}

RemoveAll Method

Syntax *ComponentRange*.**RemoveAll**

Description The RemoveAll method removes all Component objects from the ComponentRange, emptying the range.

Example The following example adds four Components to a ComponentRange and displays the contents of the range. Then the range is emptied using the RemoveAll method.

```
Private Sub Main()
    ' Add two Reports and two Scenarios to the document
    ' before trying this example
    '
    ' Dimension the variables
    Dim igxComponents As Components
    Dim Report1 As Component
    Dim Report2 As Component
    Dim Scenario1 As Component
    Dim Scenario2 As Component
    Dim igxCompRange1 As ComponentRange
    ' Get the Document's Components object
    Set igxComponents = ActiveDocument.Components
    Set igxCompRange1 = ActiveDocument.MakeComponentRange
    ' Add components to igxCompRange1
    igxCompRange1.Add igxComponents.Item(1)
    igxCompRange1.Add igxComponents.Item(2)
    igxCompRange1.Add igxComponents.Item(3)
    igxCompRange1.Add igxComponents.Item(4)
    ' Display the intermediate result
    MsgBox "ComponentRange1 contains these components:" & Chr(13) _
        & NamesIn(igxCompRange1)
    ' Remove all components from the range
    MsgBox "Click OK to remove all components from the range."
    igxCompRange1.RemoveAll
    ' Display the result
    MsgBox "ComponentRange1 contains these components:" & Chr(13) _
        & NamesIn(igxCompRange1)
End Sub

' This function returns a string containing the names of all the
' Components in a range, formatted with carriage returns
Private Function NamesIn(Range As ComponentRange) As String
    If Range.Count > 0 Then
        Dim Index As Integer, sString As String
        For Index = 1 To Range.Count
            sString = sString & Range.Item(Index).Name & Chr(13)
        Next Index
        NamesIn = Chr(13) & sString & Chr(13)
    Else
        NamesIn = "(Empty)"
    End If
End Function
```

See Also Remove method

{button ComponentRange object,JI(`igrafxrf.HLP',`ComponentRange_Object')}

RemoveRange Method

Syntax *ComponentRange*.**RemoveRange**(*Range* As ComponentRange)

Description The RemoveRange method removes Component objects from one ComponentRange based on the contents of another ComponentRange. The contents of the two ComponentRange objects are compared. If there are any Component objects in common, they are removed from the specified ComponentRange. The *Range* argument specifies which ComponentRange to use for comparison (objects are not removed from the range supplied as the argument).

Error Supplying the *Range* argument with an invalid ComponentRange produces an error. Use error trapping if your code could potentially supply the *Range* argument with an invalid ComponentRange object.

Example The following example uses the RemoveRange method to remove Components from a ComponentRange based on the contents of another ComponentRange, supplied as the argument. The first range is filled with all four Component objects. The second range is filled with just the last two. After using RemoveRange, the first range then contains just the first two components.

```
Private Sub Main()
    ' Add two Reports and two Scenarios to the document
    ' before trying this example
    '
    ' Dimension the variables
    Dim igxComponents As Components
    Dim Report1 As Component
    Dim Report2 As Component
    Dim Scenario1 As Component
    Dim Scenario2 As Component
    Dim igxCompRange1 As ComponentRange
    Dim igxCompRange2 As ComponentRange
    ' Get the Document's Components object
    Set igxComponents = ActiveDocument.Components
    Set igxCompRange1 = ActiveDocument.MakeComponentRange
    Set igxCompRange2 = ActiveDocument.MakeComponentRange
    ' Add components to igxCompRange1
    igxCompRange1.Add igxComponents.Item(1)
    igxCompRange1.Add igxComponents.Item(2)
    igxCompRange1.Add igxComponents.Item(3)
    igxCompRange1.Add igxComponents.Item(4)
    ' Add components to igxCompRange2
    igxCompRange2.Add igxComponents.Item(3)
    igxCompRange2.Add igxComponents.Item(4)
    ' Display the intermediate result
    MsgBox "ComponentRange1 contains these components:" & Chr(13) _
        & NamesIn(igxCompRange1) & Chr(13) _
        & "ComponentRange2 containes these components:" & Chr(13) _
        & NamesIn(igxCompRange2)
    ' Remove components from the first range based on the second
    MsgBox "Click OK to use RemoveRange"
    igxCompRange1.RemoveRange igxCompRange2
    ' Display the result
    MsgBox "ComponentRange1 contains these components:" & Chr(13) _
        & NamesIn(igxCompRange1) & Chr(13) _
        & "ComponentRange2 containes these components:" & Chr(13) _
```

```
        & NamesIn(igxCompRange2)
End Sub

' This function returns the names of all the components in a range
' separated with carriage returns. This function is used many times
' in the Message Boxes in the Main() subroutine.
Private Function NamesIn(Range As ComponentRange) As String
    Dim Index As Integer
    Dim sString As String
    For Index = 1 To Range.Count
        sString = sString & Range.Item(Index).Name & Chr(13)
    Next Index
    NamesIn = Chr(13) & sString & Chr(13)
End Function
```

See Also

Remove method

RemoveAll method

{button ComponentRange object,JI(`igrafxrf.HLP',`ComponentRange_Object')}

ConnectorLine Object

Topic Under Construction!!!

NEED TO ADD DESCRIPTIONS AND ILLUSTRATION OF ALL THE ARGUMENTS FOR ADDCONNECTOR, RECONNECT, ETC.

The ConnectorLine object allows the developer to work with iGrafx Professional connector lines. Connector lines are the lines that connect shapes within diagrams, and have many associated properties, such as the type of the connector, line color, arrows, crossovers, etc. The ConnectorLine object also has a number of methods and events for using and manipulating connector lines.

Programmatic issues about connectors are discussed in the topics about each of the ConnectorLine objects's properties, methods, and events, and in the iGrafx System Developer's Guide. Basic information about connectors, what they are, how to use them, etc., is presented in the iGrafx Professional User's Guide.

For information about where the ConnectorLine object fits in the API object hierarchy, see the topic iGrafx API Object Hierarchy.

Properties, Methods, and Events

All of the properties, methods, and events for the ConnectorLine object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	AfterAttach
Connector1	GetRoutePoints	AfterDetach
Connector2	ReconnectDestination	BeforeAttach
ConnectorFormat	ReconnectSource	BeforeDetach
CrossOverSize	ReverseEnds	
CrossOverType	RouteLine	
Destination		
DestinationArrowColor		
DestinationArrowSize		
DestinationArrowStyle		
DestinationDirection		
DiagramObject		
InputConnectorLines		
LineColor		
LineStyle		
LineWidth		
OutputConnectorLines		
Parent		
PermanentConnectorLine		
RepeatDestinationArrow		
Rounding		
Routing		
Source		
SourceArrowColor		
SourceArrowSize		
SourceArrowStyle		
SourceDirection		

TextObjects
UseConnectors

Related Topics

DiagramObject object
iGrafx API Object Hierarchy

AfterAttach Event

Syntax **Private Sub** *ConnectorLine*_**AfterAttach** (*Source* As DiagramObject, *Destination* As DiagramObject)

Description Two situations cause attachment events to occur:

- When a line is drawn to a shape and the line “snaps” to the shape
- When a line is moved off of one connect point on a shape and moved to another connect point, either on the same shape or a different shape.

The AfterAttach event occurs after a connector line is attached to a connect point. Custom code can be written within this event procedure to perform any desired actions. Some possible actions that could be taken in response to this event are:

- Updating the connectivity information for the model.
- Updating data associated with the connector line based on the line being attached to a particular shape or shape type.
- Changing the appearance of the connector line based on the type of shape it gets attached to.
- Changing the appearance or data associated with the attached DiagramObject.

The *Source* and *Destination* parameters provide access to the source and destination DiagramObject objects to which the connector line is attached.

Example The following example sets up a AfterAttach event that changes the colors of the shapes and the connector line involved in the attach. The Main( ) subroutine adds two shapes to the diagram ready to be attached by the user.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnector As ConnectorLine
    ' Add two shapes to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
    ' Inform the user that the event is ready
    MsgBox "AfterAttach event is ready. Try connecting the " _
       & "shapes with a connector line."
End Sub

Private Sub AnyConnector_AfterAttach(ByVal Source As IGrafx2.DiagramObject,
ByVal Destination As IGrafx2.DiagramObject)
    ' Change the colors of the shapes and connector line involved
    AnyConnector.LineColor = vbGreen
    Source.Shape.FillColor = vbGreen
    Destination.Shape.FillColor = vbGreen
End Sub
```

See Also BeforeAttach event

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

AfterDetach Event

Syntax **Private Sub** *ConnectorLine*_**AfterDetach**(*DetachedFrom* As DiagramObject)

Description The AfterDetach event occurs after a connector line is detached from a DiagramObject object. Custom code can be written within this event procedure to perform any desired actions.

This event could be useful for checking such things as shape dependencies and issuing appropriate messages to the user. Other actions could be such things as changing various properties to indicate an unconnected shape, for instance, or prompting the user for some type of action.

The *DetachedFrom* parameter provides access to the DiagramObject object from which the connector was detached.

Example The following example implements the AnyConnector AfterDetach event. If the connector line is detached from either shape in the diagram, a text message is added to which ever shape was detached.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxConnector As ConnectorLine
   ' Add two shapes to the diagram
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(3000, 3000)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(6000, 3000)
   ' Add a connector line
   Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
   ' Inform the user that the event is ready
   MsgBox "AfterDetach event is ready. Try detaching the connector " _
       & "line from either shape."
End Sub

Private Sub AnyConnector_AfterDetach(ByVal DetachedFrom As _
IGrafx2.DiagramObject)
    DetachedFrom.Shape.Text = "Connector detached from this shape"
End Sub
```

See Also BeforeDetach event

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

BeforeAttach Event

Syntax **Private Sub** *ConnectorLine* **_BeforeAttach**(*Source* As DiagramObject, *Destination* As DiagramObject, *CancelAttach* As Boolean)

Description The BeforeAttach event occurs before a connector line is attached to a connect point. Custom code can be written within this event procedure to perform any desired actions. Two situations cause attachment events to occur:

- When a line is drawn to a shape and the line “snaps” to the shape
- When a line is moved off of one connect point on a shape and moved to another connect point, either on the same shape or a different shape.

The *Source* parameter contains the DiagramObject (typically a shape) from which the connector line is drawn. The *Destination* parameter contains the DiagramObject (typically a shape) to which the connector line is drawn. You can cancel the attachment by setting the *CancelAttach* parameter to True.

Example The following example asks the user to confirm any attach actions. If the user answers No, then the connector line is not attached to the shape.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnector As ConnectorLine
    ' Add two shapes to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(3000, 3000)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(6000, 3000)
    igxShape1.DiagramObject.ObjectName = "Shape1"
    igxShape2.DiagramObject.ObjectName = "Shape2"
    ' Inform the user that the event is ready
    MsgBox "BeforeAttach event is ready." & _
    "  Try connecting the shapes with a connector line."
End Sub

Private Sub AnyConnector_BeforeAttach(ByVal Source As _
IGrafx2.DiagramObject, ByVal Destination As _
IGrafx2.DiagramObject, CancelAttach As Boolean)
    If MsgBox("Allow attachment of " & Source.ObjectName & _
    " to " & Destination.ObjectName & "?", vbYesNo) = vbNo Then
        CancelAttach = True
    End If
End Sub
```

See Also AfterAttach event

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

BeforeDetach Event

Syntax **Private Sub** *ConnectorLine* **_BeforeDetach**(*DetachFrom* As DiagramObject, *CancelDetach* As Boolean)

Description The BeforeDetach event occurs before a connector line is detached from a DiagramObject object, and so, has certain similarities to the BeforeAttach event. Custom code can be written within this event procedure to perform any desired actions.

The DetachDrom parameter contains the DiagramObject object that the connector line was detached from.

The developer can cancel the detachment by setting *CancelDetach* to True.

Example The following example implements the AnyConnector_BeforeDetach event. If a user tries to detach a connector line, the user is asked to confirm the detach.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnector As ConnectorLine
    ' Add two shapes to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(3000, 3000)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(6000, 3000)
    ' Add a connector line
    Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
    ' Inform the user that the event is ready
    MsgBox "BeforeDetach event is ready.  Try detaching the connector line."
End Sub

Private Sub AnyConnector_BeforeDetach(ByVal DetachFrom As
IGrafx2.DiagramObject, CancelDetach As Boolean)
   ' Ask the user if the detach should be canceled
   If MsgBox("Connector detaching from " & DetachFrom.ObjectName & ". Allow
   detach?", vbYesNo) = vbNo Then
        CancelDetach = True
   End If
End Sub
```

See Also AfterDetach event

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

Connector1 Property

Topic Under Construction!!!

Syntax *ConnectorLine*.**Connector1**

Data Type Integer (read/write)

Description The Connector1 property specifies the location on the connector line of the first off-page connector indicator (the one extending from the source shape). The value of the property is an index into the Points collection that stores the point locations that describe the line.

The following diagram illustrates how this property works.

Notes Off page indicators cannot reside on the end points of a line. Also, when Connector1 is moved to a different point, Connector2 is moved as well, so that they are always one point apart on the line. This will limit the points that can actually be used with the Connector1 property. If a connector line has only 4 route points, the Connector1 and Connector2 values must be 2 and 3—the two middle points. A Connector line must have more than four route points in order to have multiple choices for placement of Connector1 or Connector2.

Example The following example

This was our experimental code. It should not be included as is.-JSB

```
'Dimension variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
'Add two shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(3000, 3000)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(14000, 1000)
igxShape1.DiagramObject.ObjectName = "Shape1"
igxShape2.DiagramObject.ObjectName = "Shape2"
Set igxConnector =
ActiveDiagram.DiagramObjects.AddConnectorLine(ixRouteCurved, , igxShape1,
ixDirEast, , , , igxShape2, ixDirWest)
'Set igxConnector = ActiveDiagram.DiagramObjects.Item(4).ConnectorLine
MsgBox "true next"
igxConnector.UseConnectors = True
MsgBox "False next"
igxConnector.UseConnectors = False
Set igxPoints = igxConnector.GetRoutePoints
For Index = 2 To igxPoints.Count - 1
    igxConnector.Connector1 = Index
    MsgBox "Click OK for the next position."
Next Index
```

See Also Connector2 property

UseConnectors property

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

Connector2 Property

Topic Under Construction!!!

Syntax *ConnectorLine*.**Connector2**

Data Type Integer (read/write)

Description The Connector2 property specifies the location on the connector line of the second off-page connector indicator (the one extending from the destination shape). The value of the property is an index into the Points array that stores the point locations that describe the line.

Refer to the Connector1 property for an illustration of how this property works.

Notes Off page indicators cannot reside on the end points of a line. Also, when Connector2 is moved to a different point, Connector1 is moved as well, so that they are always one point apart on the line. This will limit the route points that can actually be used with the Connector2 property. If a connector line has only 4 route points, the Connector1 and Connector2 values must be 2 and 3—the two middle points. A connector line must have more than four route points in order to have multiple choices for placement of Connector1 or Connector2.

Example The following example

See Also Connector1 property

UseConnectors property

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

ConnectorFormat Property

Syntax *ConnectorLine*.**ConnectorFormat**

Data Type ConnectorFormat object (read-only, See Object Properties)

Description The ConnectorFormat property returns a ConnectorFormat object for the specified ConnectorLine object. The ConnectorFormat object provides control of the characteristics of the connector line and any arrows or crossovers associated with the connector line.

All of the properties available through the ConnectorFormat object are also available through the ConnectorLine object. The advantage to using the ConnectorFormat object is that it can be assigned to any ConnectorLine object as a whole. Therefore, you can create a formatting style that covers many properties, and assign it to any connector line.

Example The following example creates three shapes on the active diagram. Connector lines are added to connect the three shapes. Then, using a With statement, the formatting for connector line 1 is established in a ConnectorFormat object, defining red lines and blue arrows. Finally, the ConnectorFormat object for connector line 1 is assigned to connector line 2.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxConnLine2 As ConnectorLine
Dim igxConnFmt As ConnectorFormat
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape on the active diagram, 10 inches away
' so it is on a different page
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
Set igxShape3 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 3, 1440 * 4, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Draw a connector line between shapes 2 and 3
Set igxConnLine2 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape3, _
    ixDirNorth, ixConnectRelativeToShape)
' Get the ConnectorFormat object
With igxConnLine1.ConnectorFormat
    ' Set the ConnectorFormat properties
    .DestinationArrowFormat.Color = vbBlue
    .DestinationArrowFormat.Size = 3
    .LineFormat.Color = vbRed
    .LineFormat.Width = 3
    .RepeatDestinationArrow = True
```

```
End With
' Assign connector format from Connector 1 to Connector 2
igxConnLine2.ConnectorFormat = igxConnLine1.ConnectorFormat
```

See Also

ConnectorFormat object

iGrafx API Object Hierarchy

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

CrossOverSize Property

Syntax *ConnectorLine*.**CrossOverSize**

Data Type Integer (read/write)

Description The CrossOverSize property specifies the width of the crossover gap (the point where the connector line crosses another connector line) for a specified ConnectorLine object. This value is used for all crossover points on the connector line. Valid values for this property are 1, 2, or 3, with 1 creating the smallest gap and 3 the largest.

The CrossOverSize property is ignored if CrossOverType = ixCrossLine. The CrossOverSize property is used for all other values of the CrossOverType property.

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the same property of the ConnectorFormat object, refer to the topics listed in the See Also section).

Example The following example creates three shapes on the active diagram and connects the shapes with right angle connector lines that cross each other. The CrossOverSize and Type properties are then set for ConnectorLine 2.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxConnLine2 As ConnectorLine
Dim igxConnFmt As ConnectorFormat
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Create a third shape, 2 inches to the right and 3 inches
' below Shape 1
Set igxShape3 = igxDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
    ixDirSouth, ixConnectRelativeToShape)
' Draw a connector line between shapes 2 and 3
Set igxConnLine2 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
    ixDirNorth, ixConnectRelativeToShape, , , igxShape3, _
    ixDirNorth, ixConnectRelativeToShape)
' Set the Crossover properties for ConnectorLine 2 to Square
' and 1 (smallest)
igxConnLine2.CrossOverType = ixSquare
igxConnLine2.CrossOverSize = 1
```

See Also

CrossOverType property

ConnectorFormat property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

CrossOverType Property

Syntax *ConnectorLine*.**CrossOverType**

Data Type IxCrossOverType enumerated constant (read/write)

Description The CrossOverType property specifies the type of crossover to use for the specified ConnectorLine object. This value is used for all crossover points on the connector line.

The use of a crossover type depends on whether the connector line is above or below the connector line that is being crossed. Refer to the Description column of the table for the IxCrossOverType constants.

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the same property of the ConnectorFormat object, refer to the topics listed in the See Also section).

The IxCrossOverType constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant	Description
0	ixCrossLine	The connector line crosses other lines as if on the same level. This setting is valid no matter whether the line is above or below the line or lines being crossed.
1	ixOverLine	The connector line displays an arc-shaped (half circle) crossover when it crosses any line that is below it. The crossover type is not used when the designated connector line is below a line that crosses it.
2	ixBreakLine	The connector line displays a break on each side of the line being crossed when it crosses lines that are below it. The crossover type is not used when the designated connector line is below a line that crosses it.
3	ixSquare	The connector line displays a square-shaped crossover when it crosses any line that is below it. The crossover type is not used when the designated connector line is below a line that crosses it.
4	ixTriangle	The connector line displays a triangular -shaped crossover when it crosses any line that is below it. The crossover type is not used when the designated connector line is below a line that crosses it.

For more information about connectors and crossover types, refer to the iGrafx Professional User's Guide.

Example The following example creates three shapes on the active diagram and connects the shapes with right angle connector lines that cross each other. ConnectorLine 2 then has its crossover type and size set to ixCrossLine and 1, respectively. Then a For loop with a Select statement is used to cycle through each crossover type, displaying a message each time after the crossover type is changed. Note that because ConnectorLine 2 is above ConnectorLine 1, the BreakLine type is not used. As a test, set the CrossOverType property of ConnectorLine 1 to ixBreakLine.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
```

```
Dim igxShape3 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxConnLine2 As ConnectorLine
Dim igxConnFmt As ConnectorFormat
Dim iCount As Integer
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Create a third shape, 2 inches to the right and 3 inches
' below Shape 1
Set igxShape3 = igxDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 4, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
    ixDirSouth, ixConnectRelativeToShape)
' Draw a connector line between shapes 2 and 3
Set igxConnLine2 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
    ixDirNorth, ixConnectRelativeToShape, , , igxShape3, _
    ixDirNorth, ixConnectRelativeToShape)
' Set Connector 2 crossover size to 1 and intial type to CrossLine
igxConnLine2.CrossOverSize = 1
igxConnLine2.CrossOverType = ixCrossLine
MsgBox "View the state of the diagram"
For iCount = 1 To 5
   Select Case igxConnLine2.CrossOverType
       Case ixCrossLine:
          igxConnLine2.CrossOverType = ixOverLine
          MsgBox ("CrossOverType changed to " & _
              igxConnLine2.CrossOverType)
       Case ixOverLine:
          igxConnLine2.CrossOverType = ixBreakLine
          MsgBox ("CrossOverType changed to " & _
              igxConnLine2.CrossOverType)
       Case ixBreakLine:
          igxConnLine2.CrossOverType = ixTriangle
          MsgBox ("CrossOverType changed to " & _
              igxConnLine2.CrossOverType)
       Case ixTriangle:
          igxConnLine2.CrossOverType = ixSquare
          MsgBox ("CrossOverType changed to " & _
              igxConnLine2.CrossOverType)
       Case ixSquare:
          igxConnLine2.CrossOverType = ixCrossLine
          MsgBox ("CrossOverType changed to " & _
              igxConnLine2.CrossOverType)
    End Select
```

```
    Next iCount
```

See Also

CrossOverSize property

ConnectorFormat property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

Destination Property

Syntax *ConnectorLine*.**Destination**

Data Type DiagramObject object (read-only, See Object Properties)

Description The Destination property returns the DiagramObject object that is the destination of the specified ConnectorLine object. You can use the DiagramObject.Type property to determine the type of the destination object.

Example The following example reports whether the destination of the connector line is a shape.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxDestination As DiagramObject
' Add two shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Add a connector line to connect the shapes
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Get the destination object of the connector line
Set igxDestination = igxConnector.Destination
' Report whether the destination is a shape
If (igxDestination.Type = ixObjectShape) Then
    MsgBox "The destination of the connector line is a shape."
Else
    MsgBox "The destination of the connector line is not a shape."
End If
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

DestinationArrowColor Property

Syntax *ConnectorLine*.**DestinationArrowColor**

Data Type Color (read/write)

Description The DestinationArrowColor property specifies the color of the arrow at the destination end of the connector line. You can specify the color using any method that is valid in Visual Basic programming (refer to your Visual Basic programming documentation). This property is ignored if the SourceArrowStyle property is set to zero (no arrow).

To set color values for this property, you can use either the VB color constants (vbRed, vbGreen, etc.) or you can use the VB RGB function.

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the equivalent property of the ConnectorFormat object, refer to the topics listed in the See Also section).

Example The following example consists of two parts: a public subroutine called MyTest, and an event subroutine for the BeforeClick event for a Shape object. The MyTest subroutine creates two shapes on the active diagram and draws a connector line between them. It then sets source and destination arrow styles for the connector line, and then turns each shape into a VBA control using the `Diagram.CreateVbaControl` method. The second part is to write code in an event procedure for Shape 2, the destination shape, for the BeforeClick event. The code in this event changes the destination arrow color randomly every time Shape 2 is single clicked (selected).

```
Public Sub MyTest()

' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
     (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
     ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
     ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
' Create VBA controls for the two shapes
For Each DiagramObject In ActiveDiagram.DiagramObjects
   If (DiagramObject.IsVbaControl = False) Then
       DiagramObject.CreateVbaControl
   End If
Next DiagramObject
```

```
Private Sub Shape2_BeforeClick(ByVal X As Double, ByVal Y As Double, Cancel As
Boolean)
    Shape2.InputConnectorLines(1).ConnectorLine.DestinationArrowColor _
        = RGB(Rnd(1) * 255, Rnd(1) * 255, Rnd(1) * 255)
End Sub
```

See Also

ConnectorFormat property

SourceArrowColor property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

DestinationArrowSize Property

Syntax *ConnectorLine*.**DestinationArrowSize**

Data Type IxArrowSize enumerated constant (read/write)

Description The DestinationArrowSize property specifies the size of the arrow at the destination shape end of the connector line. The arrow sizes are roughly directly proportional (uses the same scaling) to the available sizes for the connector line. This property is ignored if the SourceArrowStyle property is set to zero (no arrow).

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the equivalent property of the ConnectorFormat object, refer to the topics listed in the See Also section).

The IxArrowSize constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
-2	ixAutomatic
1	ixVerySmall
2	ixSmall
3	ixMedium
4	ixLarge
5	ixVeryLarge

Example The following example creates two shapes on the active diagram and draws a connector line between them. The connector line given arrow styles on both the source and destination ends, and the destination arrow size set to Automatic. Then a For loop containing a With statement cycles through all the available arrow sizes for the destination arrow, and displays a message box indicating which arrow size is currently assigned.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iCount As Integer
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right of Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow3
igxConnLine1.SourceArrowStyle = ixArrow10
igxConnLine1.DestinationArrowSize = ixAutomatic
```

```
' Cycle Connector 1 destination arrow size through
' all possible values
For iCount = 1 To 6
   Select Case igxConnLine1.DestinationArrowSize
       Case ixAutomatic:
          igxConnLine1.DestinationArrowSize = ixVerySmall
          MsgBox "Destination Arrow Size changed to Very Small"
       Case ixVerySmall:
          igxConnLine1.DestinationArrowSize = ixSmall
          MsgBox "Destination Arrow Size changed to Small"
       Case ixSmall:
          igxConnLine1.DestinationArrowSize = ixMedium
          MsgBox "Destination Arrow Size changed to Medium"
       Case ixMedium:
          igxConnLine1.DestinationArrowSize = ixLarge
          MsgBox "Destination Arrow Size changed to Large"
       Case ixLarge:
          igxConnLine1.DestinationArrowSize = ixVeryLarge
          MsgBox "Destination Arrow Size changed to Very Large"
       Case ixVeryLarge:
          igxConnLine1.DestinationArrowSize = ixAutomatic
          MsgBox "Destination Arrow Size changed to Automatic"
   End Select
Next iCount
```

See Also

ConnectorFormat property

SourceArrowSize property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

DestinationArrowStyle Property

Syntax *ConnectorLine*.**DestinationArrowStyle**

Data Type IxArrowStyle enumerated constant (read/write)

Description The DestinationArrowStyle property specifies the type of arrowhead to use at the end of the connector line; that is the end of the connector that attaches to the destination shape. Valid values for this property are defined by the IxArrowStyle constant, and take the form: ixArrow0 through ixArrow55. The choices of arrow styles are best viewed through the user interface in the Format Line dialog. To display this dialog, you can do one of the following:

- Draw a connector between two shapes on a diagram. Select the connector line and then right click and select Format from the menu. Go to the Arrows and Crossovers tab.
- Select a connector line, go to the menus and choose Format—Line and Border, and then go to the Arrows and Crossovers tab.

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the equivalent property of the ConnectorFormat object, refer to the topics listed in the See Also section).

The IxArrowStyle constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixArrowNone
1-55	IxArrow1-55

Example The following example creates a connector line, and then changes the style of the source and destination arrows.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iCount As Integer
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow1
igxConnLine1.SourceArrowStyle = ixArrow10
```

See Also

SourceArrowStyle property

ReverseEnds method

ConnectorFormat property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

DestinationDirection Property

Syntax *ConnectorLine*.**DestinationDirection**

Data Type IxDirection enumerated constant (read-only)

Description The DestinationDirection property returns the direction from which the specified ConnectorLine object enters the destination DiagramObject object.

The IxDirection constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
1	ixDirNorth
2	ixDirEast
3	ixDirSouth
4	ixDirWest

For more information about connector lines, refer to the iGrafx Professional User's Guide.

Example The following example creates two shapes on the active diagram that have a connector line drawn between them of type Direct, attached to the east side of the source shape and the west side of the destination shape. It uses the DestinationDirection property to determine whether the connector line is attached on the north "side" of the shape. If not, then the ReconnectDestination method is called to attach the connector on the north side, and the Routing property is used to change the routing type to RightAngle.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
    ixDirEast, ixConnectRelativeToShape)
' If Shape 2 connection not on North, change it and set
' routing to RightAngle
If (igxConnLine1.DestinationDirection <> ixDirNorth) Then
    Call igxConnLine1.ReconnectDestination _
        (DestinationShape:=igxShape2, DestDir:=ixDirNorth)
    igxConnLine1.Routing = ixRouteRightAngle
End If
```

See Also ReconnectDestination method

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

DiagramObject Property

Syntax *ConnectorLine*.**DiagramObject**

Data Type DiagramObject object (read-only, See Object Properties)

Description A ConnectorLine is a DiagramObject. The DiagramObject property returns the ConnectorLine object's "Extender", which is the DiagramObject object associated with the connector line. Several properties and methods that are common to all objects in the diagram are at the DiagramObject level; for example, location and position properties.

If you are familiar with object-oriented terminology, you can think of the DiagramObject as the base class for the ConnectorLine object (and the base class for other objects including Shape, TextGraphic, Department, and OleObject).

Example The following example connects two shapes with connector line. It uses attributes of the connector line DiagramObject property to create a text graphic that labels the connector line.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
' Add two shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 6, 1440)
' Add a connector line to connect the shapes
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Set the object name of the connector line
igxConnector.DiagramObject.ObjectName = "Connector1"
' With block provides shorthand for the DiagramObject property
With igxConnector.DiagramObject
    ActiveDiagram.DiagramObjects.AddTextObject .CenterX - 500, _
    .CenterY + 100, , , .ObjectName
End With
MsgBox "Click OK to continue."
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

GetRoutePoints Method

Syntax *ConnectorLine*.**GetRoutePoints** As Points

Description The GetRoutePoints method returns the Points collection that contains the coordinate points that defines the connector line. This method must be assigned to a variable of type Points.

Once the Points collection has been obtained with this method, you can use the properties and methods of the Point and Points objects to manipulate the connector line. If you are going to make changes to the connector line by adjusting, adding, or removing points, be aware that there may be some additional adjustments you need to make, especially if the connector line is an off-page connector. Also, after you have made changes to one or more Point object, you need to use the RouteLine method to redraw the connector line.

For additional information regarding issues related to off-page connectors, refer to the OffPageConnectorFormat object.

Example The following example gets the Points collection from a connector line. It then iterates through points and labels each one with a text graphic. (BUG WITH THIS ???)

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
' Add three shapes, and connect the outer two. The connector
' routes around the middle shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Get the connector's Points collection
Set igxPoints = igxConnector.GetRoutePoints
MsgBox "Click OK to number the route points."
' Label all the route points
For Index = 1 To igxPoints.Count
    ActiveDiagram.DiagramObjects.AddTextObject igxPoints _
   .Item(Index).X, igxPoints.Item(Index).Y, , , Str(Index)
Next Index
' Pause for the user
MsgBox "Click OK to continue"
```

See Also RouteLine method

Point object

Points object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

InputConnectorLines Property

Syntax *ConnectorLine*.**InputConnectorLines**

Data Type ObjectRange object (read-only, See Object Properties)

Description The InputConnectorLines property returns an ObjectRange object that contains all of the ConnectorLine objects that are inputs of the specified ConnectorLine. Currently, lines cannot be connected to other lines using Visual Basic, but it can be done from the user interface. This requires switching on a special option by clicking the following check box:

Tools Menu->Options->Connector Lines Tab->Allow lines to connect to other lines

See Also ObjectRange object

iGrafx API Object Hierarchy

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

LineColor Property

Syntax *ConnectorLine*.**LineColor**

Data Type Color (read/write)

Description The LineColor property specifies the color of the connector line. The property is ignored if the IxLineStyle property is set to ixLineNone. You can specify the color using any method that is valid in Visual Basic programming (refer to your Visual Basic programming documentation).

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the same property of the ConnectorFormat object, refer to the topics listed in the See Also section).

Example The following example changes LineColor, LineStyle, and LineWidth properties of a connector line.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
'Add two shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Change the color, style, and width properties of the line
MsgBox "Click OK to change the color, style, and width of the line."
igxConnector.LineColor = vbGreen
igxConnector.LineStyle = ixLineDashed
igxConnector.LineWidth = 60
MsgBox "Click OK to continue"
```

See Also ConnectorFormat property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

LineStyle Property

Syntax *ConnectorLine*.**LineStyle**

Data Type IxLineStyle enumerated constant (read/write)

Description The LineStyle property specifies the style of the line used to draw the connector line. Line styles are solid, dashed, dotted, etc.

If the LineType property is set to ixLineNone, then this property is ignored. The LineColor property controls the color of the lines. The width of the line used to draw a graphic is controlled by the LineWidth property. For information about the available line styles, refer to the iGrafx Professional User's Guide.

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the same property of the ConnectorFormat object, refer to the topics listed in the See Also section).

The IxLineStyle constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
-2	ixLineNone
0	ixLineNormal
1	ixLineDashed
2	ixLineDotted
3	ixLineDashDot
4	ixLineDashDotDot

Example The following example changes LineColor, LineStyle, and LineWidth properties of a connector line.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
' Add two shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Change the color, style, and width properties of the line
MsgBox "Click OK to change the color, style, and width of the line."
igxConnector.LineColor = vbGreen
igxConnector.LineStyle = ixLineDashed
igxConnector.LineWidth = 60
MsgBox "Click OK to continue"
```

See Also ConnectorFormat property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

LineWidth Property

Syntax *ConnectorLine*.**LineWidth**

Data Type Integer (read/write)

Description The LineWidth property specifies the width of the line used to draw a connector line. This property is ignored if the LineStyle property is set to ixLineNone.

Valid values for this property are specified in Twips, and can be between 0 and 100. This contrasts with the user interface, where line width values are specified in points (1 point = 1/72 inch = 20 twips). A value of zero creates a very fine hairline. A value of 20 creates a one point line, 40 a two point line, 60 a three point line, etc.

Allowing the programmer to specify the line width in twips provides for finer control of the line; for instance, you could specify a 2.5 point line (50 twips) or a 2.25 point line (45 twips). The point value represented in the user interface is rounded; for instance, a 50 twip line rounds to 3 in the user interface, and a 49 twip line rounds to 2 in the user interface. This rounding does not affect the actual value you set. However, be aware that a user can change a value that you set by using the Lines and Borders dialog.

This functionality is also contained in the ConnectorFormat object. The line properties at the ConnectorLine object level have the same precedence as those at the ConnectorFormat object level. That is, whichever object sets a line property most recently is the one that is used.

For information about differences between using this property versus the same property of the ConnectorFormat object, refer to the topics listed in the See Also section).

Example The following example changes LineColor, LineStyle, and LineWidth properties of a connector line.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
' Add two shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Add a connector line
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Change the color, style, and width properties of the line
MsgBox "Click OK to change the color, style, and width of the line."
igxConnector.LineColor = vbGreen
igxConnector.LineStyle = ixLineDashed
igxConnector.LineWidth = 60
MsgBox "Click OK to continue"
```

See Also ConnectorFormat property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

OutputConnectorLines Property

Syntax *ConnectorLine*.**OutputConnectorLines**

Data Type ObjectRange object (read-only, See Object Properties)

Description The OutputConnectorLines property returns an ObjectRange object that contains all the ConnectorLine objects that are outputs of the specified ConnectorLine. Currently, lines cannot be connected to other lines using Visual Basic, but it can be done from the user interface. This requires switching on a special option by clicking the following check box:

Tools Menu->Options->Connector Lines Tab->Allow lines to connect to other lines

See Also ObjectRange object

iGrafx API Object Hierarchy

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

PermanentConnectorLine Property

Syntax *ConnectorLine*.**PermanentConnectorLine**

Data Type ConnectorLine object (read-only, See Object Properties)

Description The PermanentConnectorLine property returns a ConnectorLine object. The purpose of this property is to provide a means of holding on to the object an AnyControl is pointing at after an event is over. Since the AnyConnector property is only valid inside of events, it cannot be used to set or maintain ConnectorLine variables that exist outside of the event. To solve this problem, PermanentConnectorLine is used to return a permanent version of the ConnectorLine object that is referenced by the AnyConnector property.

The AnyControl objects are special VBA controls that are only valid during an event; these objects dynamically point at the "active" object that is triggering the event. The PermanentConnectorLine property is used to “grab” the specific object the AnyControl is pointing at so that it can be used (or accessed) once the event is over.

As an example, consider the following event procedure written for the AnyConnector_Select event.

```
Private Sub AnyConnector_Select()
    Set MyConnector = AnyConnector
End Sub
```

If the variable MyConnector is a global variable of type ConnectorLine, then within the Select event you can set MyConnector to the Connector object that is currently active. However, if you try to use MyConnector after the event is over, it returns an error because an event is not in progress. Since you set MyConnector to the AnyControl, your variable is pointing at the AnyControl that is dynamically pointing at the active object, which is Nothing outside of an event.

If your intent is to hold on to the specific connector line that the AnyConnector control is pointing at inside the event, then you need to use the PermanentConnectorLine property. This property gives you a Connector object that is valid after the event is over (outside of the event). The change to your code is as follows (MyConnector is a global variable of type ConnectorLine):

```
Private Sub AnyConnector_Select()
    Set MyConnector = AnyConnector.PermanentConnectorLine
End Sub
```

Example The following keeps a module variable always set to the last new connector line added to the diagram. This is accomplished by setting a PermanentConnectorLine object in the AnyConnectorLine_New event. The Main( ) subroutine uses the variable to change the visual attributes of the connector line.

```
' Dimension module variable
Public igxConnector As ConnectorLine

Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   ' Add two shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
   ' Add a connector line
   ActiveDiagram.DiagramObjects.AddConnectorLine _
```

```
        , , igxShape1, ixDirEast, , , , igxShape2, ixDirWest
    ' Change the color, style, and width properties of the
    ' last added connector line
    MsgBox "Click OK to change the style of the line."
    igxConnector.LineColor = vbGreen
    igxConnector.LineStyle = ixLineDashed
    igxConnector.LineWidth = 60
    MsgBox "Click OK to continue"
End Sub

Private Sub AnyConnector_New()
    ' This event sets the connector variable to the last
    ' new connector line added to the diagram
    Set igxConnector = AnyConnector.PermanentConnectorLine
End Sub
```

See Also

AnyControls object

iGrafx API Object Hierarchy

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

ReconnectDestination Method

Topic Under Construction!!!

Syntax *ConnectorLine*.**ReconnectDestination**(*DestinationShape* As Shape, [*RouteFlag* As IxRouteFlag **=** ixRouteFlagFindEdge], [*DestDir* As IxDirection], *DestConnectType* As IxConnectType **=** ixConnectRelativeToShape], [*DestX* As Long = -1], [*DestY* As Long = -1])

Description The ReconnectDestination method changes (or reconnects) the direction from which a connector line attaches to its destination shape. For instance, if a connector line is attached on the north side (top) of the destination shape, you can use this method to change the direction (the side) from which the connector line enters the shape. The direction is defined by the IxDirection constant (see the description of the method's arguments).

The *DestinationShape* argument specifies the Shape object to connect to as the destination.

The *RouteFlag* argument specifies whether the connector line is routed to the actual edge of the shape, or just to the bounding box of the shape. The IxRouteFlag constant defines the valid values.

Value	Name of Constant
0	ixRouteFlagFindEdge
1	ixRouteFlagDontFindEdge

The *DestDir* argument specifies the direction from which the connector line is attached to the destination shape. The IxDirection constant defines the valid values.

Value	Name of Constant
1	ixDirNorth
2	ixDirEast
3	IxDirSouth
4	ixDirWest

The *DestConnectType* argument ??? The IxConnectType constant defines the valid values.

Value	Name of Constant
0	ixConnectRelativeToShape
1	ixConnectAbsoluteFromTopLeft

Example The following example creates two shapes on the active diagram that have a connector line drawn between them of type Direct, attached to the east side of the source shape and the west side of the destination shape. It uses the DestinationDirection property to determine whether the connector line is attached on the north "side" of the shape. If not, then the ReconnectDestination method is called to attach the connector on the north side, and the Routing property is used to change the routing type to RightAngle.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
```

```
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
    ixDirEast, ixConnectRelativeToShape)
' If Shape 2 connection not on North, change it and set
' routing to RightAngle
If (igxConnLine1.DestinationDirection <> ixDirNorth) Then
    Call igxConnLine1.ReconnectDestination _
        (DestinationShape:= igxShape2, DestDir:= ixDirNorth)
    igxConnLine1.Routing = ixRouteRightAngle
End If
```

See Also DestinationDirection property

ReconnectSource method

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

ReconnectSource Method

Topic Under Construction!!!

Syntax

ConnectorLine.**ReconnectSource** *SourceShape* As Shape, [*RouteFlag* As IxRouteFlag **=** ixRouteFlagFindEdge], [*SourceDir* As IxDirection], *SourceConnectType* As IxConnectType **=** ixConnectRelativeToShape], [*SourceX* As Long = -1], [*SourceY* As Long = -1]

Description

The ReconnectSource method changes (or reconnects) the direction from which a connector line attaches to its source shape. For instance, if a connector line is attached on the north side (top) of the source shape, you can use this method to change the direction (the side) from which the connector line leaves the shape. The direction is defined by the IxDirection constant (see the description of the method's arguments).

The *SourceShape* argument specifies the Shape object to connect to as the source.

The *RouteFlag* argument specifies whether the connector line is routed to the actual edge of the shape, or just to the bounding box of the shape. The IxRouteFlag constant defines the valid values.

Value	Name of Constant
0	ixRouteFlagFindEdge
1	ixRouteFlagDontFindEdge

The *SourceDir* argument specifies the direction from which the connector line is attached to the source shape. The IxDirection constant defines the valid values.

Value	Name of Constant
1	ixDirNorth
2	ixDirEast
3	IxDirSouth
4	ixDirWest

The *SourceConnectType* argument ??? The IxConnectType constant defines the valid values.

Value	Name of Constant
0	ixConnectRelativeToShape
1	ixConnectAbsoluteFromTopLeft

Example

The following example creates two shapes on the active diagram that have a connector line drawn between them of type Direct, attached to the east side of the source shape and the west side of the destination shape. It uses the SourceDirection property to determine whether the connector line is attached on the north "side" of the shape. If not, then the ReconnectSource method is called to attach the connector on the north side, and the Routing property is used to change the routing type to RightAngle.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
```

```
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
    ixDirEast, ixConnectRelativeToShape)
' If Shape 1 connection not on North, change it and set
' routing to RightAngle
If (igxConnLine1.SourceDirection <> ixDirNorth) Then
    Call igxConnLine1.ReconnectSource _
        (SourceShape:= igxShape1, SourceDir:= ixDirNorth)
    igxConnLine1.Routing = ixRouteRightAngle
End If
```

See Also SourceDirection property

ReconnectDestination method

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

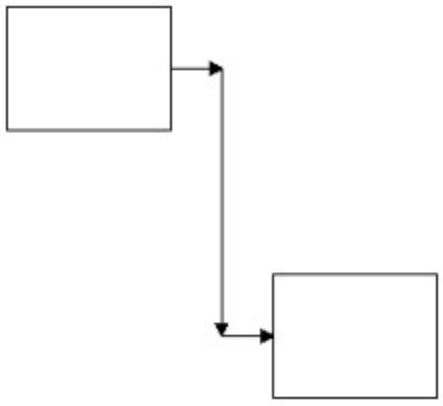
RepeatDestinationArrow Property

Syntax *ConnectorLine*.**RepeatDestinationArrow**[= {**True | False**}]

Data Type Boolean (read/write)

Description The RepeatDestinationArrow property specifies whether to draw the end arrow type at the ends of all line segments (value = True), or just at the end of the final line segment (value = False). This property is ignored if the DestinationArrowStyle property is set to zero (no arrow).

The following illustration shows destination arrows being repeated on a connector line.



Example The following example uses the RepeatDestinationArrow property to display multiple arrows on a connector line.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
' Add three shapes, and connect the outer two. The connector
' routes around the middle shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
MsgBox "Click OK to repeat the destination arrow."
igxConnector.RepeatDestinationArrow = True
MsgBox "Click OK to continue."
```

See Also DestinationArrowStyle property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

ReverseEnds Method

Syntax *ConnectorLine*.**ReverseEnds**

Description The ReverseEnds method flips, or reverses, the arrow types that have been applied to a connector line.

Example The following example creates two shapes on the active diagram, and routes and connector line of type “Direct” between them. It then sets the source and destination arrow styles, and then applies the ReverseEnds method to exchange the source and destination arrow styles. To see the result, run the code first with the last line commented out (the call to the ReverseEnds method). Then remove the comment from the last line and run the code again.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iCount As Integer
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow1
igxConnLine1.SourceArrowStyle = ixArrow10
igxConnLine1.ReverseEnds
```

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

RouteLine Method

Syntax *ConnectorLine*.**RouteLine**(*pPoints* As Points, [RouteFlag As IxRouteFlag = ixRouteFlagFindEdge])

Description The RouteLine method redraws the specified ConnectorLine object using the Points collection designated by the *pPoints* argument. The *RouteFlag* argument specifies whether the connector line should snap to the actual edge of the shapes, or to the bounding box.

For more information about the RouteFlag argument, refer to the discussion of the ConnectorLine object.

Example The following example gets the Points collection from a connector line, modifies it, and reapplies the Points collection using the RouteLine method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
' Add three shapes, and connect the outer two. The connector
' routes around the middle shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Get the connector's Points collection
Set igxPoints = igxConnector.GetRoutePoints
MsgBox "Click OK to move two of the points"
' Move two of the points
igxPoints.Item(3).Y = igxPoints.Item(3).Y - 500
igxPoints.Item(4).Y = igxPoints.Item(3).Y - 500
' Reapply the Points collection to the connector line
igxConnector.RouteLine igxPoints
' Pause for the user
MsgBox "Click OK to continue"
```

See Also Points object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

Rounding Property

Syntax *ConnectorLine*.**Rounding**

Data Type Long (read/write)

Description The Rounding property specifies the amount of rounding to apply to all corners of the specified connector line. The units are in twips (1440 twips = 1 inch.) For instance, to apply rounded corners that have a radius a 1/4 inch, use a value of 360. This works as long as each line segment is at least 1/4 inch long.

If you apply a rounding factor that is larger than 50% of any of the line segments, then the rounding for the corner or corners that cannot support the rounding value is limited to 50% of the shortest segment. However, all corners whose two segments can support the rounding value are rounded to the specified setting.

In addition, if the line is changed so that the line segments get longer, the current value of the property is re-applied; that is, the value of the property itself does not change if its full value cannot be applied to the specified connector line intially (see the example).

Example The following example creates three shapes in a row, and connects the two end shapes with a right-angle connector line. Then, 1/4 inch rounding is applied to the connector line. As you can see, all the segments can support 1/4 inch rounding. Next, the rounding is increased to 3/4 inch. This time, not all of the segments can round to 3/4 inch, so those that can use it, and those that cannot use 50% of the shortest line segment. This behavior is further illustrated by moving the connected shapes down slightly, and increasing the size of the middle shape so that the connector line must grow.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
' Add three shapes, and connect the outer two. The connector
' routes around the middle shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
MsgBox "Click OK to apply rounding to the connector line."
' Apply 1/4 inch rounded corners
igxConnector.Rounding = 360
' Pause for the user
MsgBox "Connector rounded 1/4 inch. Now increase the rounding " _
    & "to 3/4 inch."
' Increase the rounding
igxConnector.Rounding = 1080
MsgBox "View the result. Now move the connected shapes down."
' Move the connected shapes down and apart
igxShape1.DiagramObject.CenterX = 1440
igxShape1.DiagramObject.CenterY = 1440 * 2
igxShape2.DiagramObject.CenterX = 1440 * 6
igxShape2.DiagramObject.CenterY = 1440 * 2
igxShape3.DiagramObject.Height = 1440 * 2.5
igxShape3.DiagramObject.Width = 1440 * 2
igxShape3.DiagramObject.CenterX = 1440 * 3.5
```

```
igxShape3.DiagramObject.CenterY = 1440 * 2
MsgBox "View the result."
```

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

Routing Property

Syntax *ConnectorLine*.**Routing**

Data Type IxRouteType enumerated constant (read/write)

Description The Routing property specifies the type of routing to apply to the specified ConnectorLine object. If you require more information about routing, refer to the iGrafx Professional User's Guide.

The IxRouteType constant defines the valid values for this property, which are listed in the following table. ???

Value	Name of Constant	Description
0	ixRouteDirect	
1	ixRouteRightAngle	
2	ixRouteCurved	
3	ixRouteOrgChart	
4	ixRouteCauseAndEffect	
5	ixRouteLightningBolt	

Example The following example creates three shapes on the active diagram and connects the shapes with right angle connector lines that cross each other. ConnectorLine 2 then has its crossover type and size set to ixCrossLine and 1, respectively. Then a For loop with a Select statement is used to cycle through each crossover type, displaying a message each time after the crossover type is changed.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iCount As Integer
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Cycle Connector 1 routing type through all possible values
For iCount = 1 To 6
   Select Case igxConnLine1.Routing
       Case ixRouteDirect:
          igxConnLine1.Routing = ixRouteRightAngle
          MsgBox ("Routing Type changed to " & _
```

```
            igxConnLine1.Routing)
         Case ixRouteRightAngle:
            igxConnLine1.Routing = ixRouteCurved
            MsgBox ("Routing Type changed to " & _
                igxConnLine1.Routing)
         Case ixRouteCurved:
            igxConnLine1.Routing = ixRouteOrgChart
            MsgBox ("Routing Type changed to " & _
                igxConnLine1.Routing)
         Case ixRouteOrgChart:
            igxConnLine1.Routing = ixRouteCauseAndEffect
            MsgBox ("Routing Type changed to " & _
                igxConnLine1.Routing)
         Case ixRouteCauseAndEffect:
            igxConnLine1.Routing = ixRouteLightningBolt
            MsgBox ("Routing Type changed to " & _
                igxConnLine1.Routing)
         Case ixRouteLightningBolt:
            igxConnLine1.Routing = ixRouteDirect
            MsgBox ("Routing Type changed to " & _
                igxConnLine1.Routing)
      End Select
   Next iCount
```

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

Source Property

Syntax *ConnectorLine*.**Source**

Data Type DiagramObject object (read-only, See Object Properties)

Description The Source property returns the DiagramObject object that is the source of the specified ConnectorLine object. The source object is typically a shape.

Example The following example uses the Source property to access the source shape for the connector line, and change it's color.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector As ConnectorLine
Dim igxPoints As Points
' Add three shapes, and connect the outer two. The connector
' routes around the middle shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Change the color of the source shape
MsgBox "Click OK to change the color of the connector line's source."
igxConnector.Source.Shape.FillColor = vbGreen
' Pause for the user
MsgBox "Click OK to continue"
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

SourceDirection Property

Syntax *ConnectorLine*.**SourceDirection**

Data Type IxDirection enumerated constant (read-only)

Description The SourceDirection property returns the direction from which the specified ConnectorLine object leaves the source DiagramObject object. The source object typically is a shape, but does not have to be a shape.

The IxDirection constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
1	ixDirNorth
2	ixDirEast
3	ixDirSouth
4	ixDirWest

For more information about connectors and crossover types, refer to the iGrafx Professional User's Guide.

Example The following example creates two shapes on the active diagram that have a connector line drawn between them of type Direct, attached to the east side of the source shape and the west side of the destination shape. It uses the SourceDirection property to determine whether the connector line is attached on the north "side" of the shape. If not, then the ReconnectSource method is called to attach the connector on the north side, and the Routing property is used to change the routing type to RightAngle.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
   (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
    ixDirEast, ixConnectRelativeToShape)
' If Shape 1 connection not on North, change it and set
' routing to RightAngle
If (igxConnLine1.SourceDirection <> ixDirNorth) Then
    Call igxConnLine1.ReconnectSource _
        (SourceShape:= igxShape1, SourceDir:= ixDirNorth)
    igxConnLine1.Routing = ixRouteRightAngle
End If
```

See Also

DestinationDirection property

ReconnectSource method

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

SourceArrowColor Property

Syntax *ConnectorLine*.**SourceArrowColor**

Data Type Color (read/write)

Description The SourceArrowColor property specifies the color of the arrow at the source shape end of the connector line. You can specify the color using any method that is valid in Visual Basic programming (refer to your Visual Basic programming documentation). This property is ignored if the SourceArrowStyle property is set to zero (no arrow).

To set color values for this property, you can use either the VB color constants (vbRed, vbGreen, etc.) or you can use the VB RGB function.

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the equivalent property of the ConnectorFormat object, refer to the topics listed in the See Also section).

Example Refer to the Example section of the DestinationArrowColor property for a complete example. The basic form of setting this property is shown below, where it is assumed that the `igxConnLine1` variable previously has been set to a ConnectorLine object.

```
igxConnLine1.SourceArrowColor = vbRed
OR
igxConnLine1.SourceArrowColor = RGB(255, 0, 0)
```

See Also DestinationArrowColor property

ConnectorFormat property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

SourceArrowSize Property

Syntax *ConnectorLine*.**SourceArrowSize**

Data Type IxArrowSize enumerated constant (read/write)

Description The SourceArrowSize property specifies the size of the arrow at the source shape end of the connector line. The arrow sizes are roughly directly proportional (uses the same scaling) to the available sizes for the connector line. This property is ignored if the SourceArrowStyle property is set to zero (no arrow).

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the equivalent property of the ConnectorFormat object, refer to the topics listed in the See Also section).

The IxArrowSize constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
-2	ixAutomatic
1	ixVerySmall
2	ixSmall
3	ixMedium
4	ixLarge
5	ixVeryLarge

Example Refer to the Example section of the DestinationArrowSize property for a complete example. To run the code, replace all references to the DestinationArrowSize property with the SourceArrowSize property.

See Also DestinationArrowSize property

ConnectorFormat property

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

SourceArrowStyle Property

Syntax *ConnectorLine*.**SourceArrowStyle**

Data Type IxArrowStyle enumerated constant (read/write)

Description The SourceArrowStyle property specifies the type of arrowhead to use at the beginning of the connector line; that is the end of the connector that attaches to the source shape. Valid values for this property are defined by the IxArrowStyle constant, and take the form: ixArrow0 through ixArrow55. The choices of arrow styles are best viewed through the user interface in the Format Line dialog. To display this dialog, you can do one of the following:

- Draw a connector between two shapes on a diagram. Select the connector line and then right click and select Format from the menu. Go to the Arrows and Crossovers tab.
- Select a connector line, go to the menus and choose Format—Line and Border, and then go to the Arrows and Crossovers tab.

Note that this property is also available through the ConnectorFormat object. For information about differences between using this property versus the equivalent property of the ConnectorFormat object, refer to the topics listed in the See Also section).

The IxArrowStyle constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixArrowNone
1-55	IxArrow1-55

Example The following example creates two shapes on the active diagram with a connector line drawn between them. It then sets both the SourceArrowStyle and DestinationArrowStyle properties.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine1 As ConnectorLine
Dim iCount As Integer
' Get the active diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the first shape on the active diagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape _
    (1440, 1440, Application.ShapeLibraries.Item(1).Item(1))
' Create a second shape, 4 inches to the right and 1 inch
' below Shape 1
Set igxShape2 = igxDiagram.DiagramObjects.AddShape _
   (1440 * 5, 1440 * 2, Application.ShapeLibraries.Item(1).Item(1))
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = igxDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, ixRouteFlagFindEdge, igxShape1, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
    ixDirWest, ixConnectRelativeToShape)
' Set Connector 1 arrow types
igxConnLine1.DestinationArrowStyle = ixArrow1
igxConnLine1.SourceArrowStyle = ixArrow10
```

See Also

DestinationArrowStyle property

ConnectorFormat property

ReverseEnds method

ConnectorFormat object

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

TextObjects Property

Syntax *ConnectorLine*.**TextObjects**

Data Type ObjectRange object (read-only, See Object Properties)

Description The TextObjects property returns an ObjectRange object that tells you which, if any, TextGraphicObject objects are attached to the connector line.

Example The following example creates two shapes and a connector line. It also creates a text graphic object, and attaches it to the connector line. Then the TextObjects property is used to access all the text graphic objects attached to the connector line, and changes the fill color.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxText As TextGraphicObject
Dim igxConnector As ConnectorLine
Dim igxRange As ObjectRange
' Add three shapes, and connect them
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add a text object
Set igxText = ActiveDiagram.DiagramObjects.AddTextObject _
   (3500, 800, , , "Connector")
' Attach the text to the connector line
igxText.AttachTo igxConnector.DiagramObject
' Get the object range of text objects from the connector
Set igxRange = igxConnector.TextObjects
' Change the fill color of all text objects in the range
MsgBox "Click OK to change the fill color of any text attached " _
   & "to the connector line."
igxRange.FillFormat.FillColor = vbGreen
' Pause for the user
MsgBox "Click OK to continue"
```

See Also ObjectRange object

iGrafx API Object Hierarchy

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

UseConnectors Property

Syntax *ConnectorLine*.**UseConnectors**[= {**True | False**}]

Data Type Boolean (read/write)

Description The UseConnectors property specifies whether a connector line is a normal connector line (value = False) or an off-page connector line (value = True). This property affects individual connector lines, so you can use it to set up off page connectors on a “per connector line” basis. To have iGrafx Professional automatically create off page connector lines for all connector lines in a diagram that cross page boundaries, use the OffPageConnectorFormat.AutomaticConnectors property (the parent object of OffPageConnectorFormat is Diagram).

Once you set this property to True, the specified connector line becomes an off-page connector line, shown below.

The two lines are still one ConnectorLine object, and are controlled by the ConnectorLine object’s properties, methods, and events. However, the connector indicators are controlled by the OffPageConnectorFormat object. The OffPageConnectorFormat properties also control many aspects of off-page connectors that are applied on a per diagram basis.

To get the points that describe the line segments of the connector line, use the GetRoutePoints method, which returns a Points collection. You can use the Points collection to alter the connector line, and then use the RouteLine method to redraw the connector line.

Example The following example creates a connector line that connects shapes on two different pages. The UseConnectors property is used to turn off page connectors on and off.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
' Add two shapes, and connect them
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 8, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Zoom out a little
ActiveDiagram.Views.Item(1).DiagramView.ZoomPercentage = 75
' Change the UseConnectors property
igxConnector.UseConnectors = False
MsgBox "UseConnectors = False"
igxConnector.UseConnectors = True
MsgBox "UseConnectors = True"
igxConnector.UseConnectors = False
' Pause for the user
MsgBox "UseConnectors = False"
```

See Also GetRoutePoints method

RouteLine method

OffPageConnectorFormat object (see AutomaticConnectors property)

{button ConnectorLine object,JI(`igrafxrf.HLP',`ConnectorLine_Object')}

Entity Object

The Entity object is a marker that represents an execution point in an iDiagram. An entity travels from shape to shape within one or more diagrams by following connector lines.

An entity is a generalized element: essentially, it can represent anything you want—or at least, anything you can model within iGrafx Professional. For example, if you wanted to diagram the process of customers entering a bank to perform one or more transactions, the Entity objects in your diagram could represent the customers, the bank employees, even the cash itself. What an entity represents is up to the diagram's creator. You can create numerous entities within a diagram, have them run in sequence or simultaneously, and have them start at various locations.

When a entity runs, it follows four rules that govern its behavior:

1. Shapes along the path being traversed are queried, or "asked," whether the entity can enter. If the answer is "No," the entity stops.
2. Enter the shape. The EntityAccept event fires.
3. Check the shape for links to other diagrams or shapes, and if any exist, execute them. The following criteria are applied to executing a link:
   - While there is a link to another diagram
   - Execute link to next shape
   - Go back to 1
4. Execute VBA code that is behind shape.

If an entity has a link and the link designates a start point, the entity jumps into the appropriate diagram at that start point (could be the same, or a different diagram). If the entity has no start point designated in the link, the entity jumps to the first created object in the diagram. If the entity has a link to a diagram that does not have any objects, the diagram is not activated and the entity does not jump.

The following example can be used to try all of the entity-related events. The "Main" subroutine creates a diagram with four connected shapes, and an entity in the first shape. To run this example, place the "Main" subroutine in a Diagram-level project. Then, place all of the event subroutines in a Document-level project (ThisDocument, for example). Run the Main subroutine to create the diagram, then go to the iGrafx Professional interface and Run the entity (using the Entity Manager, or the Run button on the iDiagram toolbar). To test the EntitiesAbort event, press the Stop button or the Esc key during execution.

```
Sub Main()
  ' Dimension the variables
  Dim igxShape1 As Shape
  Dim igxShape2 As Shape
  Dim igxConnLine As ConnectorLine
  Dim igxEntity As Entity
  ' Create 2 shapes in the diagram and connect them
  Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
  Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
     (1440 * 3, 1440)
  ' Add connector line
  Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
     (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
     ixDirEast, ixConnectRelativeToShape, , , igxShape2, _
     ixDirWest, ixConnectRelativeToShape)
  ' Create an entity in the first shape
  Set igxEntity = ActiveDocument.Entities.Add("MyEntity", igxShape1)
  ' Add a third shape and connect it to shape 2
  Set igxShape1 = igxShape2
  Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
```

```
      (1440 * 3, 1440 * 3)
  ' Add connector line
  Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
      (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
      ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
      ixDirNorth, ixConnectRelativeToShape)
  ' Add a third shape and connect it to shape 2
  Set igxShape1 = igxShape2
  Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
      (1440, 1440 * 3)
  ' Add connector line
  Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
      (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
      ixDirWest, ixConnectRelativeToShape, , , igxShape2, _
      ixDirEast, ixConnectRelativeToShape)
  ' Display message box
  MsgBox "Open the Entity Manager dialog and click the Run button."
End Sub
```

Place all of the following event subroutines in a Document-level project, such as ThisDocument.

```
Private Sub AnyShape_EntitiesAbort(ByVal Error As Long)
  Me.AnyShape.FillColor = vbRed
  For iCount = 0 To 3000
      DoEvents
  Next iCount
  MsgBox "The EntityAbort event was triggered."
End Sub

---------------------------------

Private Sub AnyShape_EntitiesFinished()
  Me.AnyShape.FillColor = vbBlack
  For iCount = 0 To 3000
      DoEvents
  Next iCount
  MsgBox "All entities finished. Shape turns black when " _
        & "EntititesFinished event" & Chr(13) & "has completed " _
        & " for that Shape "
End Sub

---------------------------------

Private Sub AnyShape_EntitiesStart()
  Me.AnyShape.FillColor = vbGreen
  Me.AnyShape.Text = "Starting"
  For iCount = 0 To 3000
      DoEvents
  Next iCount
  MsgBox "The EntitiesStart event was fired."
```

```
End Sub

---------------------------------

Private Sub AnyShape_EntityAccept(AcceptEntity As Boolean, ByVal Entity As
IGrafx2.IXEntity)
  Me.AnyShape.FillColor = vbBlue
  Me.AnyShape.Text = "Accepted"
  For iCount = 0 To 3000
      DoEvents
  Next iCount
  MsgBox "The " & Entity.Name & " entity was accepted."
End Sub

---------------------------------

Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
  Me.AnyShape.FillColor = vbCyan
  Me.AnyShape.Text = "Executing"
  Entity.Size = ixEntityLarge
  For iCount = 0 To 3000
      DoEvents
  Next iCount
  Entity.Size = ixEntityNormal
End Sub

---------------------------------

Private Sub AnyShape_EntityInitiate(ByVal Entity As IGrafx2.IXEntity)
  Me.AnyShape.FillColor = vbMagenta
  Me.AnyShape.Text = "Initiate"
  For iCount = 0 To 3000
      DoEvents
  Next iCount
  MsgBox "The EntityInitiate event has fired."
End Sub

---------------------------------

Private Sub AnyShape_EntityLeave(ByVal Entity As IGrafx2.IXEntity)
  Me.AnyShape.FillColor = vbWhite
  Me.AnyShape.Text = "Leaving"
  For iCount = 0 To 3000
      DoEvents
  Next iCount
  MsgBox "EntityLeave event done for " & AnyShape.ObjectName
End Sub

---------------------------------
```

```
Private Sub AnyShape_EntityStep(ByVal Entity As IGrafx2.IXEntity)
  Me.AnyShape.FillColor = vbYellow
  Me.AnyShape.Text = "Step event is active"
  For iCount = 0 To 3000
     DoEvents
  Next iCount
  Me.AnyShape.Text = ""
End Sub
```

Properties, Methods, and Events

All of the properties, methods, and events for the Entity object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	AccumulateData	
BorderColor	Delete	
Color	Goto	
CurrentPath	Pause	
CustomDataValues	Resume	
Delay	Run	
LastPath	SelectPath	
Location	SetGraphic	
Name	SetStandardGraphic	
Parent	Stop	
Size		

AccumulateData Method

Syntax *Entity*.**AccumulateData**(*DiagramObject* As DiagramObject)

Description The AccumulateData method causes the Entity to accumulate the CustomDataValue's of the specified DiagramObject. Each time the AccumulateData method is executed, the Entity accumulates the CustomDataValue objects of specified DiagramObject, and stores the result in the Entity's own CustomDataValues collection. The result is based on the AccumulationMethod of each corresponding CustomDataDefinition at the Document level.

The DiagramObject argument specifies the DiagramObject from which to accumulate more custom data. It is useful to use the AccumulateData method within AnyShape.Entity events, and use the AnyShape.DiagramObject object for the *DiagramObject* argument (see the example below.)

Example The following example creates a simple diagram and an Entity. As the Entity progresses through the diagram, it's accumulated data is displayed.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxShape3 As Shape
    Dim igxConnector1 As ConnectorLine
    Dim igxConnector2 As ConnectorLine
    Dim igxEntity As Entity
    ' Add objects to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
    Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
    igxShape1.Text = "$10"
    igxShape2.Text = "$20"
    igxShape3.Text = "$30"
    ' Add connector lines
    Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape1, ixDirEest, , , , igxShape2, _
       ixDirWast)
    Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape3, _
       ixDirWest)
    ' Add a CustomDataDefinition to the document
    ActiveDocument.CustomDataDefinitions.Add _
        "Cost", ixCustomDataFormatCurrencyBase
    ' Set each Shape's Cost value
    igxShape1.DiagramObject.CustomDataValues.Item _
       ("Cost", ixCustomDataCurrency).Value = 10
    igxShape2.DiagramObject.CustomDataValues.Item _
       ("Cost", ixCustomDataCurrency).Value = 20
    igxShape3.DiagramObject.CustomDataValues.Item _
       ("Cost", ixCustomDataCurrency).Value = 30
    ' Add an Entity to the diagram
    ActiveDocument.Entities.Add "MyEntity", igxShape1
    ' Run the entity
    ActiveDocument.Entities.Item(1).Run
End Sub
```

```
Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
    ' Accumulate data from the AnyShape
    Entity.AccumulateData AnyShape.DiagramObject
    ' Display the accumulation result so far
    MsgBox "Entity accumulated cost: " & Entity.CustomDataValues _
        .Item("Cost", ixCustomDataCurrency)
End Sub
```

See Also

CustomDataDefinition.AccumulationMethod property

DiagramObject.CustomDataValues property

Document.CustomDataDefinitions property

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

BorderColor Property

Syntax *Entity*.**BorderColor**

Data Type Long (read/write)

Description The BorderColor property specifies the BorderColor of the Entity object. You can use any valid Visual Basic method for specifying a color, such as the RGB function or one of the VB color constants.

Example The following example changes the border color of an Entity object from its default to red.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxEntity As Entity
' Add two shapes, and connect them
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add an entity to the first shape
ActiveDocument.Entities.Add "TestEntity", igxShape1
Set igxEntity = ActiveDocument.Entities.Item(1)
' Make the Entity appear large
igxEntity.Size = ixEntityLarge
MsgBox "Click OK to change the border color of the Entity marker."
igxEntity.BorderColor = vbRed
' Pause for the user
MsgBox "Click OK to continue."
```

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

CurrentPath Property

Syntax *Entity*.**CurrentPath**

Data Type Path object (read-only, See Object Properties)

Description The CurrentPath property returns the Path object that defines the current path on which the specified entity is traveling.

Example The following example uses an Entity's CurrentPath property to access the destination shape of the Entity.

```
Private igxPath As Path

Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnector1 As ConnectorLine
    Dim igxEntity As Entity
    ' Turn off the "Finished" message box
    ShowFinished = False
    ' Add two shapes
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
    ' Add a connector
    Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
    ' Add an entity
    ActiveDocument.Entities.Add "TestEntity", igxShape1
    Set igxEntity = ActiveDocument.Entities.Item(1)
    ' Run the entity
    MsgBox "Click OK to run the entity."
    igxEntity.Run
    DoEvents
    ' Pause for the user
    MsgBox "Click OK to continue."
End Sub

Private Sub AnyShape_EntityLeave(ByVal Entity As IGrafx2.IXEntity)
   On Error GoTo Done
   Set igxPath = Entity.CurrentPath
   AnyShape.PermanentShape.Text = "Entity has left here"
   igxPath.Destination.Text = "Entity will end up here"
Done:
   MsgBox "Entity has reached the end of it's path"
End Sub
```

See Also Path object

iGrafx API Object Hierarchy

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

CustomDataValues Property

Syntax *Entity*.**CustomDataValues**

Data Type CustomDataValues collection object (read-only, See Object Properties)

Description The CustomDataValues property returns the CustomDataValues object for the specified entity.

Example The following example uses the Entity object's CustomDataValues collection to access a custom data value. The value is a text value used to store the name of the shape each time the Entity enters a shape.

```
' Dimension module variable
Private igxValue As CustomDataValue

Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnector1 As ConnectorLine
    Dim igxEntity As Entity
    ' Turn off the "Finished" message box
    ShowFinished = False
    ' Add two shapes
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
    igxShape2.DiagramObject.ObjectName = "Shape 2"
    ' Add a connector
    Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
    ' Add an entity
    ActiveDocument.Entities.Add "TestEntity", igxShape1
    Set igxEntity = ActiveDocument.Entities.Item(1)
    ActiveDocument.CustomDataDefinitions.Add "MyData", _
       ixCustomDataFormatTextBase
    Set igxValue = igxEntity.CustomDataValues.Item(1, ixCustomDataText)
    igxEntity.Run
    DoEvents
    MsgBox "The last shape the entity reached was: " & igxValue.Value
End Sub

Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
    igxValue.Value = AnyShape.PermanentDiagramObject.ObjectName
End Sub
```

See Also CustomDataValues object

iGrafx API Object Hierarchy

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

Delay Property

Syntax *Entity*.**Delay**

Data Type Long (read/write)

Description The Delay property specifies the current delay value for the entity. The amount of delay determines how many seconds the entity stays at its current shape. The default value is 1. This property has no meaning when the iDiagram containing the entity is not running.

Example The following example sets up an Entity with delay. Events change the color of each shape that the Entity occupies so the Entity's progress can be seen.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxShape3 As Shape
    Dim igxConnector1 As ConnectorLine
    Dim igxConnector2 As ConnectorLine
    Dim igxEntity As Entity
    ' Turn off the "Finished" message box
    ShowFinished = False
    ' Add shapes to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
    Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
    ' Add connectors to the diagram
    Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
    Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
    ' Add an entity
    ActiveDocument.Entities.Add "TestEntity", igxShape1
    Set igxEntity = ActiveDocument.Entities.Item(1)
    ' Set the delay to 1000
    igxEntity.Delay = 1000
    MsgBox "To test the delay, return to the diagram and click Run."
End Sub

Private Sub AnyShape_EntityAccept(AcceptEntity As Boolean, ByVal Entity As
IGrafx2.IXEntity)
    AnyShape.FillColor = vbBlue
    DoEvents
End Sub

Private Sub AnyShape_EntityLeave(ByVal Entity As IGrafx2.IXEntity)
   AnyShape.FillColor = vbWhite
   DoEvents
End Sub
```

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

Goto Method

Syntax *Entity*.**Goto**(*Shape* As Shape) As Boolean

Description The Goto method of the EntityObject causes a jump to a specific object in a diagram directly rather than navigating the paths to get to the object. The *Shape* argument specifies which shape the entity goes to next.

Note This method is only valid within the Entity_Leave event.

Example The following example uses Entity events to monitor the Entity. When the Entity leaves Shape 1, the Goto method is used to make it jump to Shape 3, skipping Shape 2. Each shape the Entity enters is colored blue to show it's progress.

```
' Dimesion module variables
Private WithEvents igxShape1 As Shape
Private igxShape2 As Shape
Private igxShape3 As Shape
Private igxShape4 As Shape

Private Sub Main()
    ' Dimension the variables
    Dim igxConnector1 As ConnectorLine
    Dim igxConnector2 As ConnectorLine
    Dim igxConnector3 As ConnectorLine
    Dim igxEntity As Entity
    ' Turn off the "Finished" message box
    ShowFinished = False
    ' Add shapes to the diagram
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
    Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
    Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 7, 1440)
    ' Add connectors to the diagram
    Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
    Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
    Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape3, ixDirEast, , , , igxShape4, ixDirWest)
    ' Add an entity
    ActiveDocument.Entities.Add "TestEntity", igxShape1
    Set igxEntity = ActiveDocument.Entities.Item(1)
    MsgBox "To test the Goto jump, return to the diagram and click Run."
End Sub

' When the entity leaves Shape 1, have it jump to Shape 3
Private Sub igxShape1_EntityLeave(ByVal Entity As IGrafx2.IXEntity)
   Entity.Goto igxShape3
End Sub

' Change the color of each shape that the Entity enters
Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
   AnyShape.FillColor = vbBlue
End Sub
```

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

LastPath Property

Syntax *Entity*.**LastPath**

Data Type Path object (read-only, See Object Properties)

Description The LastPath property returns the Path object that was last used by the specified Entity object.

Example The following example uses the LastPath property to display the previous source and destination shapes of the Entity object.

```
' Dimension module variables
Private igxPath As Path

Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxConnector1 As ConnectorLine
    Dim igxEntity As Entity
    ' Turn off the "Finished" message box
    ShowFinished = False
    ' Add two shapes
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
    ' Give the shapes meaningful object names
    igxShape1.DiagramObject.ObjectName = "Shape 1"
    igxShape2.DiagramObject.ObjectName = "Shape 2"
    ' Add a connector to the diagram
    Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
        (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
    ' Add an entity to Shape 1
    ActiveDocument.Entities.Add "TestEntity", igxShape1
    Set igxEntity = ActiveDocument.Entities.Item(1)
    ' Run the entity
    MsgBox "Click OK to run the entity."
    igxEntity.Run
    DoEvents
End Sub

Private Sub AnyShape_EntityLeave(ByVal Entity As IGrafx2.IXEntity)
    ' Skip this if there is no current path
    On Error GoTo Done
    Set igxPath = Entity.CurrentPath
Done:
    ' Use LastPath to get the previous source and destination
    ' of Entity
    MsgBox "Entity has finished.  It's last path" & Chr(13) _
        & "went from " & _
    Entity.LastPath.Source.DiagramObject.ObjectName _
        & " to " & _
    Entity.LastPath.Destination.DiagramObject.ObjectName & "."
End Sub
```

See Also

Path object

iGrafx API Object Hierarchy

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

Location Property

Syntax *Entity*.**Location**

Data Type Shape object (read-only, See Object Properties)

Description The Location property returns the Shape object in which the specified Entity object currently resides.

Example The following example reports the location of the Entity every time it enters a new shape.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxShape3 As Shape
    Dim igxConnector1 As ConnectorLine
    Dim igxConnector2 As ConnectorLine
    Dim igxEntity As Entity
    ' Turn off the "Finished" message box
    ShowFinished = False
    ' Add three shapes
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
    Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
    ' Give the shapes meaningful object names
    igxShape1.DiagramObject.ObjectName = "Shape 1"
    igxShape2.DiagramObject.ObjectName = "Shape 2"
    igxShape3.DiagramObject.ObjectName = "Shape 3"
    ' Add connector lines to the diagram
    Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
    Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
    ' Add an entity to Shape 1
    ActiveDocument.Entities.Add "TestEntity", igxShape1
    Set igxEntity = ActiveDocument.Entities.Item(1)
    ' Run the entity
    MsgBox "Click OK to run the entity."
    igxEntity.Run
    DoEvents
End Sub

Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
    MsgBox "The Entity is now in " & Entity.Location.DiagramObject.ObjectName
End Sub
```

See Also Shape object

iGrafx API Object Hierarchy

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

Pause Method

Syntax *Entity*.**Pause**

Description The Pause method pauses the specified Entity object when it is running. The Pause method is equivalent to selecting an Entity in the Entity Mananger and clicking the Pause button. No events are triggered by this method.

Example The following example has a MainSetup( ) subroutine, and two events. The MainSetup( ) subroutine sets up two shapes, a connector line, and an entity ready to be Run by the user.

The EntityAccept event changes whichever shape receives an entity to blue. In addition, the event checks which shape triggered the event. If it was Shape 2, the Pause method pauses the Entity. The user then clicks to resume the Entity, using the Resume method. The EntityExecute event changes the shape back to white.

```
Private Sub MainSetup()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxShape3 As Shape
   Dim igxConnector1 As ConnectorLine
   Dim igxConnector2 As ConnectorLine
   Dim igxEntity As Entity
   ' Turn off the "Finished" message box
   ShowFinished = False
   ' Add three shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
   Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
   ' Give the shapes meaningful object names
   igxShape1.DiagramObject.ObjectName = "Shape 1"
   igxShape2.DiagramObject.ObjectName = "Shape 2"
   igxShape3.DiagramObject.ObjectName = "Shape 3"
   ' Add connector lines to the diagram
   Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
   Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
   Set igxConnector3 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape3, ixDirSouth, , , , igxShape1, _
       ixDirSouth)
   ' Add an entity to Shape 1
   ActiveDocument.Entities.Add "TestEntity", igxShape1
   Set igxEntity = ActiveDocument.Entities.Item(1)
   ' Inform the user that the setup is complete
   MsgBox "Entity ready. Return to the diagram and click Run."
End Sub

Private Sub AnyShape_EntityAccept(AcceptEntity As Boolean, _
ByVal Entity As IGrafx2.IXEntity)
   ' Change the shape blue when the Entity enters
   AnyShape.FillColor = vbBlue
   ' Pause the Entity at Shape 2
   If AnyShape.DiagramObject.ObjectName = "Shape 2" Then
       Entity.Pause
```

```
        MsgBox "Entity Paused on Shape 2. Click OK to resume (Or click Stop)"
        Entity.Resume
    End If
    ' The following performs a timed pause of 1/2 second to slow
    ' down the progress of the Entity
    Dim StartTime As Double
    StartTime = Timer
    While (True)
        If Timer > StartTime + 0.5 Then
           Exit Sub
        End If
        DoEvents
    Wend
End Sub

' Change the shape back to white when the Entity leaves
Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
    AnyShape.FillColor = vbWhite
    DoEvents
End Sub
```

See Also

Resume method

Run method

Stop method

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

Resume Method

Syntax *Entity*.**Resume**

Description The Resume method causes the specified Entity object to resume running after it has been paused. The Resume method is equivalent to selecting an Entity in the Entity Manager and clicking the Resume button. No events are triggered by this method.

Example See the example for the Pause method.

See Also Pause method

Run method

Stop method

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

Run Method

Syntax *Entity*.**Run**

Description The Run method starts an entity running. The Run method is equivalent to selecting an Entity in the Entity Manager and clicking the Run button.

Example The following example runs an Entity object. When the entity reaches Shape 3, it is stopped.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxShape3 As Shape
   Dim igxConnector1 As ConnectorLine
   Dim igxConnector2 As ConnectorLine
   Dim igxEntity As Entity
   ' Turn off the "Finished" message box
   ShowFinished = False
   ' Add three shapes
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
   Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
   ' Give the shapes meaningful object names
   igxShape1.DiagramObject.ObjectName = "Shape 1"
   igxShape2.DiagramObject.ObjectName = "Shape 2"
   igxShape3.DiagramObject.ObjectName = "Shape 3"
   ' Add connector lines to the diagram
   Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
   Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
   ' Add an entity to Shape 1
   ActiveDocument.Entities.Add "TestEntity", igxShape1
   Set igxEntity = ActiveDocument.Entities.Item(1)
   ' Run the entity, and display actions in the Output pane
   MsgBox "Click OK to run the entity."
   igxEntity.Run
End Sub

' When the entity executes in Shape 3, stop it
Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
   If (AnyShape.DiagramObject.ObjectName = "Shape 3") Then
       Entity.Stop
       MsgBox "The Entity reached Shape 3."
   End If
End Sub
```

See Also

Pause method

Resume method

Stop method

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

SelectPath Method

Syntax *Entity*.**SelectPath**(*Path* As Integer)

Description The SelectPath method directs the Entity down a particular path from a shape. This method is only valid within the EntityExecute event, and can only be called while the Entity is at a shape and the diagram is running, otherwise it has no effect. The SelectPath method can be called at anytime before an Entity leaves the shape. If an invalid Path Number is supplied, the diagram stops and displays an error message. If the is more than one output path that an entity can travel, then the entity stops at the current shape if the SelectPath property has not been set before the Entity needs to leave the shape.???

Example The following example arranges a diagram that has two paths. Every time the entity is run, it alternates which path it takes. Color changes show the progress of the Entity object. The SelectPath method appears in the EntityExecute event, which directs the Entity to a particular path.

```
' Dimension module variable
Private Count As Integer

Private Sub Main()
    ' Dimension the variables
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxShape3 As Shape
    Dim igxConnector1 As ConnectorLine
    Dim igxConnector2 As ConnectorLine
    Dim igxConnector3 As ConnectorLine
    Dim igxEntity As Entity
    ' Turn off the "Finished" message box
    ShowFinished = False
    ' Add four shapes
    Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 2)
    Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 2)
    Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
    Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440 * 3)
    ' Give the shapes meaningful object names
    igxShape1.DiagramObject.ObjectName = "Shape 1"
    igxShape2.DiagramObject.ObjectName = "Shape 2"
    igxShape3.DiagramObject.ObjectName = "Shape 3"
    ' Add connector lines to the diagram
    Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
    Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
    Set igxConnector3 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape4, ixDirWest)
    ' Add an entity to Shape 1
    ActiveDocument.Entities.Add "TestEntity", igxShape1
    Set igxEntity = ActiveDocument.Entities.Item(1)
    ' Ask the user to Run the Entity several times
    MsgBox "Return to the diagram and click Run several times." _
       & Chr(13) & "The Entity will alternate which path it follows."
End Sub

Private Sub AnyShape_EntityAccept(AcceptEntity As Boolean, _
```

```
ByVal Entity As IGrafx2.IXEntity)
    ' Change the shape blue when the Entity enters
    AnyShape.FillColor = vbBlue
End Sub

' When the entity executes in Shape 3, stop it
Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
   ' Change the shape color back to white
   AnyShape.FillColor = vbWhite
   ' Increment the count
   Count = Count + 1
   ' Alternate which shape the Entity goes to
   If (AnyShape.DiagramObject.ObjectName = "Shape 2") Then
      If Count Mod 2 = 0 Then
          Entity.SelectPath 1
      Else
          Entity.SelectPath 2
      End If
   End If
   ' The following performs a timed pause of 1/2 second to slow
   ' down the progress of the Entity
   Dim StartTime As Double
   StartTime = Timer
   While (True)
      If (Timer > StartTime + 0.5) Then
         Exit Sub
      End If
      DoEvents
   Wend
End Sub
```

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

SetGraphic Method

Syntax *Entity*.**SetGraphic**(*GraphicName* As String, *GraphicLocation* As IxEntityGraphicLocation) As Boolean

Description The SetGraphic method sets the symbol of an Entity to a shape graphic in a ShapeLibrary. The method returns a Boolean result indicating success or failure of the method.

The *GraphicName* argument specifies the name of the graphic to use.

The *GraphicLocation* argument specifies the location from which to retreive the graphic. The method searches through all available Shape Libraries based on the *GraphicLocation* argument. The IxEntityGraphicLocation constant defines the valid values, which are listed in the following table. If set to IxGraphicFromDiagram, the graphic is retrieved from the Diagram's ShapeLibrary. If set to IxGraphicFromMediaManager, the graphic is retrieved from the MediaManager ShapeLibrary.

Value	Name of Constant
0	ixGraphicFromDiagram
1	ixGraphicFromMediaManager

Example The following example changes the graphic used to display the Entity. In this case the Airplane symbol.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxConnector3 As ConnectorLine
Dim igxEntity As Entity
' Turn off the "Finished" message box
ShowFinished = False
' Add four shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 2)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440 * 3)
' Give the shapes meaningful object names
igxShape1.DiagramObject.ObjectName = "Shape 1"
igxShape2.DiagramObject.ObjectName = "Shape 2"
igxShape3.DiagramObject.ObjectName = "Shape 3"
igxShape4.DiagramObject.ObjectName = "Shape 4"
' Add connector lines to the diagram
Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
Set igxConnector3 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape4, ixDirWest)
' Add an entity to Shape 1
ActiveDocument.Entities.Add "TestEntity", igxShape1
```

```
Set igxEntity = ActiveDocument.Entities.Item(1)
MsgBox "Click OK to change the Entity's graphic to an airplane."
' Make the Entity graphic large
igxEntity.Size = ixEntityLarge
' Set the Entity's graphic
igxEntity.SetGraphic "Airplane", ixGraphicFromMediaManager
MsgBox "Click OK to continue."
```

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

SetStandardGraphic Method

Syntax *Entity*.**SetStandardGraphic**(*Graphic* As IxEntityGraphic) As Boolean

Description The SetStandardGraphic method sets the Entity's symbol to a circle, square, diamond, or star. The method returns a Boolean result indicating success or failure of the method.

The *Graphic* argument specifies which of the standard graphical symbols to use. The IxEntityGraphic constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixEntityCircle
1	ixEntitySquare
2	ixEntityDiamond
3	ixEntityStar

Example The following example changes the Entity's graphic to a circle.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxConnector3 As ConnectorLine
Dim igxEntity As Entity
' Turn off the "Finished" message box
ShowFinished = False
' Add four shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 2)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440 * 3)
' Give the shapes meaningful object names
igxShape1.DiagramObject.ObjectName = "Shape 1"
igxShape2.DiagramObject.ObjectName = "Shape 2"
igxShape3.DiagramObject.ObjectName = "Shape 3"
' Add connector lines to the diagram
Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
Set igxConnector3 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape4, ixDirWest)
' Add an entity to Shape 1
ActiveDocument.Entities.Add "TestEntity", igxShape1
Set igxEntity = ActiveDocument.Entities.Item(1)
MsgBox "Click OK to change the Entity's graphic."
' Make the Entity graphic large
igxEntity.Size = ixEntityLarge
' Set the Entity's graphic
igxEntity.SetStandardGraphic ixEntityCircle
MsgBox "Click OK to continue."
```

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

Size Property

Syntax *Entity*.**Size**

Data Type IxEntitySize enumerated constant (read/write)

Description The Size property specifies the size of the Entity object's symbol.

The IxEntitySize constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
1	ixEntitySmall
2	ixEntityNormal
3	ixEntityLarge

Example See the example for the SetStandardGraphic property.

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

Stop Method

Syntax *Entity*.**Stop**

Description The Stop method stops the specified Entity object if it is running. The Stop method can be used any time while an entity is running. Using this method triggers the EntitiesAbort event.

Example The following example sets up three shapes and an Entity object. After the entity is run, it is stopped at Shape 3.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxShape3 As Shape
   Dim igxConnector1 As ConnectorLine
   Dim igxConnector2 As ConnectorLine
   Dim igxEntity As Entity
   ' Turn off the "Finished" message box
   ShowFinished = False
   ' Add three shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
   Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
   ' Give the shapes meaningful object names
   igxShape1.DiagramObject.ObjectName = "Shape 1"
   igxShape2.DiagramObject.ObjectName = "Shape 2"
   igxShape3.DiagramObject.ObjectName = "Shape 3"
   ' dd connector lines to the diagram
   Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
   Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
   Set igxConnector3 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteRightAngle, , igxShape3, ixDirSouth, , , , igxShape1, _
       ixDirSouth)
   ' Add an entity to Shape 1
   ActiveDocument.Entities.Add "TestEntity", igxShape1
   Set igxEntity = ActiveDocument.Entities.Item(1)
    ' Run the entity, and display actions in the Output pane
   MsgBox "Entity ready. Return to the diagram and click Run."
End Sub

' Change the shape blue when the Entity enters
Private Sub AnyShape_EntityAccept(AcceptEntity As Boolean, ByVal Entity As
IGrafx2.IXEntity)
   AnyShape.FillColor = vbBlue
   ' Stop the entity at shape 3
   If (AnyShape.DiagramObject.ObjectName = "Shape 3") Then
       Entity.Stop
       MsgBox "Entity stopped at Shape 3"
   End If
   ' The following performs a timed pause of 1/2 second to slow
   ' down the progress of the Entity
   Dim StartTime As Double
```

```
   StartTime = Timer
   While (True)
       If (Timer > StartTime + 0.5) Then
          Exit Sub
       End If
       DoEvents
   Wend
End Sub

' Change the shape back to white when the Entity leaves
Private Sub AnyShape_EntityLeave(ByVal Entity As IGrafx2.IXEntity)
    AnyShape.FillColor = vbWhite
   DoEvents
End Sub
```

See Also

Pause method

Resume method

Run method

{button Entity object,JI(`igrafxrf.HLP',`Entity_Object')}

Entities Object

The Entities object is a collection of individual Entity objects. An Entities collection is only associated with and accessible from a Document object. Its purpose is to store and provide access to the individual Entity objects.

The Entities object provides the following functionality:

- The ability to access any Entity objects that have been created for a particular Document object.
- The ability to determine how many Entity objects are in the collection.
- The ability to add a new Entity object to the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the Entities object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent		

Add Method

Syntax *Entities*.**Add**(*Name* As String, *StartShape* As Shape) As Entity

Description The Add method adds a new Entity object to the Entities collection for the specified Document object. This method allows the addition of new entities that can be displayed in a diagram. The result of the method must be assigned to a variable of type Entity.

The *Name* argument specifies an identifying name for the new entity. The name you specify cannot be the same as an existing entity, or else an error is returned.

The *StartShape* argument allows you to specify the shape where the entity should begin execution.

Example The following example adds a new Entity object. Note that the result of the method must be assigned to a variable of type Entity.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxConnector3 As ConnectorLine
Dim igxEntity As Entity
' Turn off the "Finished" message box
ShowFinished = False
' Add four shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 2)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440 * 3)
' Give the shapes meaningful object names
igxShape1.DiagramObject.ObjectName = "Shape 1"
igxShape2.DiagramObject.ObjectName = "Shape 2"
igxShape3.DiagramObject.ObjectName = "Shape 3"
' Add connector lines to the diagram
Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
Set igxConnector3 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape4, ixDirWest)
' Add an entity to Shape 1
ActiveDocument.Entities.Add "TestEntity", igxShape1
Set igxEntity = ActiveDocument.Entities.Item(1)
MsgBox "Click OK to change the Entity's graphic."
' Make the Entity graphic large
igxEntity.Size = ixEntityLarge
' Set the Entity's graphic
igxEntity.SetStandardGraphic ixEntityCircle
MsgBox "Click OK to continue."
```

{button Entities object,JI(`igrafxrf.HLP',`Entities_Object')}

Item Method

Syntax *Entities*.**Item**

Description The Item method returns the Entity object at the specified *Index* from the Entities collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Entity. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example adds two Entity objects to a branching diagram. It then sends one entity down the first path, and the other entity down the second path. The Item method is used to select which Entity object is run. In this case, Item(2), TestEntity2.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxShape2 As Shape
   Dim igxShape3 As Shape
   Dim igxConnector1 As ConnectorLine
   Dim igxConnector2 As ConnectorLine
   Dim igxConnector3 As ConnectorLine
   Dim igxEntity1 As Entity
   Dim igxEntity2 As Entity
   ' Turn off the "Finished" message box
   ShowFinished = False
   ' Add four shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 2)
   Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 2)
   Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
   Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440 * 3)
   ' Give the shapes meaningful object names
   igxShape1.DiagramObject.ObjectName = "Shape 1"
   igxShape2.DiagramObject.ObjectName = "Shape 2"
   igxShape3.DiagramObject.ObjectName = "Shape 3"
   igxShape4.DiagramObject.ObjectName = "Shape 4"
   ' Add connector lines to the diagram
   Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
   Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape3, ixDirWest)
   Set igxConnector3 = ActiveDiagram.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape2, ixDirEast, , , , igxShape4, ixDirWest)
   ' Add two entities to Shape 1
   ActiveDocument.Entities.Add "TestEntity1", igxShape1
   Set igxEntity1 = ActiveDocument.Entities.Item(1)
   ActiveDocument.Entities.Add "TestEntity2", igxShape1
   Set igxEntity2 = ActiveDocument.Entities.Item(2)
   MsgBox "Click OK to Run TestEntity2"
   ' Run the second Entity, Item(2)
   ThisDocument.Entities.Item(2).Run
End Sub

Private Sub AnyShape_EntityAccept(AcceptEntity As Boolean, ByVal Entity As
IGrafx2.IXEntity)
```

```
   ' When an entity reaches the end of the flowchart, display the
   ' appropriate message.
   If AnyShape.DiagramObject.ObjectName = "Shape 3" Then
      MsgBox Entity.Name & " has reached Shape 3."
   End If
   If AnyShape.DiagramObject.ObjectName = "Shape 4" Then
      MsgBox Entity.Name & " has reached Shape 4."
   End If
End Sub

Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
   ' If the Entity is in Shape 2, choose a path
   If AnyShape.DiagramObject.ObjectName = "Shape 2" Then
       If Entity.Name = "TestEntity1" Then
           ' Send TextEntity1 down the first path
           Entity.SelectPath 1
       Else
           ' Send TextEntity1 down the second path
           Entity.SelectPath 2
       End If
   End If
End Sub
```

{button Entities object,JI(`igrafxrf.HLP',`Entities_Object')}

EventManager Object

The EventManager object provides some control over the propagation of event messages through the system. It's only purpose is to stop an event message. This object has one property, CancelBubble. If you set CancelBubble to True, you cancel further bubbling of the current event.

For example, when you double click on a shape, that event bubbles through multiple controls. For a shape, the number of controls the event bubbles through is effectively doubled if the event is an "Extender" event. For more information on extenders, see the DiagramObject object.

The following list shows the controls (in order of arrival) that an extender event would travel through.

- The VBA control for the shape (if there is one created).
- The AnyShape control at the Diagram level.
- The AnyObject control at the Diagram level.
- The AnyShape control at the Document level.
- The AnyObject control at the Document level.
- The AnyShape control at the Document's DiagramType level.
- The AnyObject control at the Document's DiagramType level.
- The ShapeClass's shape control at the Document level.
- The Application's DiagramType AnyShape control.
- The Application's DiagramType AnyObject control.
- Any Extension projects AnyShape control
- Any Extension projects AnyObject control

If you were to set CancelBubble = True in the event handler in the VBA control for the shape, the event would stop there and would not go to all the other controls.

For more information, see Event Bubbling in the Developer's Guide.

Properties, Methods, and Events

All of the properties, methods, and events for the EventManager object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
CancelBubble		
Parent		

Related Topics

AnyControls object

DiagramObject object

CancelBubble Property

Syntax *EventManager*.**CancelBubble**[= {**True | False**}]

Data Type Boolean (read/write)

Description The CancelBubble property cancels an event bubble. It prevents further propagation of an event through the system. If set to True, the event is cancelled at that point, and no other levels of the system will hear the event. If set to False, the event bubbles to the next level normally.

Example The following example sets up a Shape_BeforeClick event at the Diagram level, and also at the Document level. When the user clicks a shape, both events are fired, one after the other. However, if the user chooses to cancel the bubble, the Document-level event never gets fired.

To try this example, put each of these events in the code window indicated. Add a shape to the diagram if necessary, and click any shape.

```
' This is the Diagram level event
' Put this in a Diagram code window
Private Sub AnyShape_BeforeClick(ByVal X As Double, ByVal Y As Double,
Cancel As Boolean)
    ' Ask the user if they want to cancel the event bubble
    If MsgBox("Diagram level event fired. Cancel bubble?", vbYesNo) _
    = vbYes Then
        EventManager.CancelBubble = True
    Else
        EventManager.CancelBubble = False
    End If
End Sub

-------------------------------------------------------

' This is the Document level event
' Put this in the ThisDocument code window
Private Sub AnyShape_BeforeClick(ByVal X As Double, ByVal Y As Double,
Cancel As Boolean)
    MsgBox "Document level event fired."
End Sub
```

{button EventManager object,JI(`igrafxrf.HLP',`EventManager_Object')}

Field Object

The field object is a display field on a DiagramObject that can show a variety of data. A field has predefined data that can be displayed, and predefined positions for placement of the field. Data that can be displayed includes the current time, date, shape notes, custom data, document file name, and the results of VBA expressions, to name a few. The placement of fields is defined by the FieldPostion property; there are a wide range of predefined positions, as well as the option for a user to define a field's location manually.

Field objects are contained within the Fields collection object, which is subordinate to DiagramObject objects.

Properties, Methods, and Events

All of the properties, methods, and events for the Field object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
FieldPosition		
FieldText		
Font		
Frozen		
HasMaximumWidth		
Hidden		
MaximumWidth		
Orientation		
Parent		
ShowDescription		
TextAlignment		
XOffset		
YOffset		

FieldPosition Property

Syntax *Field*.**FieldPosition**

Data Type IxFieldPosition enumerated constant (read/write)

Description The FieldPosition property specifies the location, within or bordering a DiagramObject, to place the specified Field.

The IxFieldPosition constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixFieldAboveLeft1
1	ixFieldAboveLeft2
2	ixFieldAboveLeft3
3	ixFieldAbove
4	ixFieldAboveRight1
5	ixFieldAboveRight2
6	ixFieldAboveRight3
7	ixFieldBelowLeft1
8	ixFieldBelowLeft2
9	ixFieldBelowLeft3
10	ixFieldBelow
11	ixFieldBelowRight1
12	ixFieldBelowRight2
13	ixFieldBelowRight3
14	ixFieldLeftTop
15	ixFieldLeft
16	ixFieldLeftBottom
17	ixFieldRightTop
18	ixFieldRight
19	ixFieldRightBottom
20	ixFieldInsideTopLeft
21	ixFieldInsideTop
22	ixFieldInsideTopRight
23	ixFieldInsideLeft
24	ixFieldInsideCenter
25	ixFieldInsideRight
26	ixFieldInsideBottomLeft
27	ixFieldInsideBottom
28	ixFieldInsideBottomRight
29	ixFieldOtherPosition

Example The following example creates a shape in the active diagram, and adds a note to the shape. Then a Note field is defined for the shape. Using the FieldPosition property, the field is then positioned at locations 0 through 9.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape As Shape
Dim igxField1 As Field
' Set igxDiagram to Diagram object
Set igxDiagram = Application.ActiveDiagram
' Create the shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440)
igxShape.Note = "TEXT FIELD"
' Add a note field to the shape just defined
Set igxField1 = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextShapeNote, "", ixFieldBelow)
' Step through positions 0 through 9
For iCount = 0 To 9
    ' Change the position
    igxField1.FieldPosition = iCount
    ' Have a message box display the position number
    MsgBox "This is field position " & iCount
Next iCount
```

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

FieldText Property

Syntax *Field*.**FieldText**

Data Type FieldText object (read-only, See Object Properties)

Description The FieldText property returns the FieldText object for the specified Field object. The FieldText object, depending on the “type” of the field text, provides access to the formatted value of the field, additional formatting controls, or access to the CustomDataDefinition object. It also allows you to call a macro.

Example The following example creates a shape and adds a shape note. Then the shape is given a field that displayes the shape note. The field text is then hidden so it is not displayed, and then the field text is displayed in a message box.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create the shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440)
' Add a note
igxShape.Note = "FIELD TEXT"
' Add a note field to the shape just defined
Set igxField1 = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextShapeNote, "", ixFieldBelow)
MsgBox "Field1 is of type: " & igxField1.FieldText.Type
' Hide the field text within the shape
igxField1.Hidden = True
' Have a message box display the field text
MsgBox igxField1.FieldText
```

See Also FieldText object

iGrafx API Object Hierarchy

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

Font Property

Syntax *Field*.**Font**

Data Type Font object (read-only, See Object Properties)

Description The Font property returns the Font object associated with the specified Field object. You use this property to change the font, font style, font size, and font color of the Field object's text.

Example The following example illustrates how to change a field's font color to red:

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create the shape on the active diagram.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440)
igxShape.Note = "Text color test"
' Add a note field to the shape just defined
Set igxField1 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextShapeNote, "", ixFieldBelow)
' Change the font to red
MsgBox "Change the font to red."
igxField1.Font.Color = vbRed
MsgBox "Click OK to continue."
```

See Also Font object

iGrafx API Object Hierarchy

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

Frozen Property

Syntax *Field*.**Frozen**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Frozen property specifies whether the field contents are frozen. If frozen, the Field object cannot be automatically updated. For instance, if a field displays the current date, the date is frozen and does not get updated the next time the diagram is opened. The Frozen property only is valid for fields that contain a date or custom data.

The Frozen property does not affect appearance attributes of the field, such as its position or font style.

Example The following example creates a Current Date field. It then freezes the field so the date cannot be updated.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a field containing the current date below the shape
Set igxField1 = igxShape.DiagramObject. _
   Fields.Add(ixFieldTextCurrentDate, "", ixFieldBelow)
' Freeze the field contents
MsgBox "Click OK to freeze the field."
igxField1.Frozen = True
' Prompt the user to try to move the text.
MsgBox "The date on this field will remain as is. It will not update."
```

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

HasMaximumWidth Property

Syntax *Field*.**HasMaximumWidth** [= {**True | False**}]

Data Type Boolean (read/write)

Description The HasMaximumWidth property specifies whether a field has a limited width. If set to True, the field has a default MaximumWidth of 1440 twips (one inch), which can then be increased or decreased by the programmer. When HasMaximumWidth is set to True, the text is wrapped and continued on the next line. If set to False, the text remains on one continuous line, with no word wrap.

The HasMaximumWidth property affects the MaximumWidth property. When HasMaximumWidth is set to True, MaximumWidth is always initially changed to 1440 twips, but can then be set to any value the programmer chooses.

When HasMaximumWidth is set to False, the MaximumWidth property is always changed to 0 (zero), though the MaximumWidth value actually has no effect, due to the HasMaximumWidth property being False.

Example The following example takes the current date field, and wraps it to a maximum width of 1440 twips.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a current date field to the shape
Set igxField1 = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextCurrentDate, "", ixFieldBelow)
MsgBox "View the diagram"
' Set the maximum width to True
igxField1.HasMaximumWidth = True
MsgBox "View the diagram"
```

See Also MaximumWidth property

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

Hidden Property

Syntax *Field*.**Hidden**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Hidden property specifies whether to hide the specified Field object. If the property is True, the field is not displayed in the diagram. If False, the field is displayed.

Example The following example creates a shape in the active diagram, and adds a date field to the shape. The field is then hidden and unhidden.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a current date field to the shape
Set igxField1 = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextCurrentDate, "", ixFieldBelow)
MsgBox "View the diagram"
' Hide the date field
MsgBox "Click OK to hide the date field"
igxField1.Hidden = True
' Unhide the date field
MsgBox "Click OK to unhide the date field"
igxField1.Hidden = False
MsgBox "View the diagram"
```

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

MaximumWidth Property

Syntax *Field*.**MaximumWidth**

Data Type Long (read/write)

Description The MaximumWidth property allows the user to set a maximum width for the specified Field object. If the HasMaximum width property is set to True, then the field text is wrapped based on the width set by this property if the text exceeds the maximum width.

The maximum width is defined in twips (1440 twips = 1 inch). Refer to the HasMaximumWidth property for more information.

Example The following example takes the current date field, and wraps it to a maximum width of 1000 twips:

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = igxDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a current date field to the shape
Set igxField1 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextCurrentDate, "", ixFieldBelow)
' Prompt the user to change maximum width of the date field
MsgBox "Set the maximum width to 720 twips (1/2 inch)"
' Define the maximum width as 720 twips
igxField1.MaximumWidth = 720
MsgBox "The maximum width is now 720 twips"
```

See Also HasMaximumWidth property

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

Orientation Property

Syntax *Field*.**Orientation**

Data Type IxOrientation enumerated constant (read/write)

Description The Orientation property specifies the orientation of the specified Field object. The rotation of the field is always relative to the starting point of 0 degrees, not the current rotation of the field.

The IxOrientation constant defines the valid values for this property, which are listed in the following table.

Note: Although the IxOrientation constant defines a rotation of 180 degrees, this value is not valid for a Field object, and so should not be used. If it is used, the field orients to 0 degrees instead (see the code example).

Value	Name of Constant
0	ixOrientation0
1	ixOrientation90
2	ixOrientation180
3	ixOrientation270

Example The following example creates a shape in the active diagram, and adds to it a current date field. The Orientation property is then used to display the text in each of the three valid orientations. Note that if you set this property to ixOrientation180, that value is intercepted and automatically replaced by the value ixOrientation0. This is verified in the sample because Case 2 of the Select statement is not entered.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a current date field to the shape
Set igxField1 = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextCurrentDate, "", ixFieldBelow)
MsgBox "View the diagram"
' Rotate the field through all possible orientations
For iCount = 0 To 3
    igxField1.Orientation = iCount
    Select Case igxField1.Orientation
        Case 0:
            MsgBox "Field oriented to 0 degrees--" _
                & "Horizontal text"
        Case 1:
            MsgBox "Field oriented to 90 degrees--" _
                & "Vertical text, reading down the page"
        Case 2:
            MsgBox "A Field oriented to 180 degrees would" _
                & " make the text upside down." & Chr(13) _
                & "This setting is changed to orient the " _
                & "text horizontally."
        Case 3:
```

```
            MsgBox "Field oriented to 270 degrees--" _
                & "Vertical text, reading up the page"
        End Select
    Next iCount
    igxField1.Orientation = ixOrientation0
    MsgBox "End of sample"
```

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

ShowDescription Property

Syntax *Field*.**ShowDescription**[= {**True | False**}]

Data Type Boolean (read/write)

Description The ShowDescription property specifies whether the descriptive name of the field is displayed in the diagram, or just the value of the field. If the property is True, the description is displayed. If False, only the value is displayed.

Example The following example example creates a shape in the active diagram, and adds to it a current date field. The description of the field is displayed and then removed.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a current date field to the shape
Set igxField1 = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextCurrentDate, "", ixFieldBelow)
MsgBox "View the diagram"
' Show the desription of the date field
MsgBox "Click OK to show the description"
igxField1.ShowDescription = True
MsgBox "Click OK to remove the description"
igxField1.ShowDescription = False
MsgBox "End of example"
```

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

TextAlignment Property

Syntax *Field*.**TextAlignment**

Data Type IxHorizontalAlignment enumerated constant (read/write)

Description The TextAlignment property specfies how the Field object is aligned horizontally within the area allocated for the field.

The IxHorizontalAlignment constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixHorizontalAlignLeft
1	ixHorizontalAlignRight
2	ixHorizontalAlignCenter

Example The following example example creates a shape in the active diagram, and adds to it a current date field. A maximum width is set, and then the text is aligned to all three possible settings.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a current date field to the shape
Set igxField1 = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextCurrentDate, "", ixFieldBelow)
MsgBox "View the diagram"
' Make the maximum width of the field 1 and 1/4 inch
igxField1.MaximumWidth = 1440 + 360
' Align the text left
igxField1.TextAlignment = ixHorizontalAlignLeft
MsgBox "Text is aligned to the left"
' Align the text right
igxField1.TextAlignment = ixHorizontalAlignRight
MsgBox "Text is aligned to the right"
' Align the text to the center
igxField1.TextAlignment = ixHorizontalAlignCenter
MsgBox "Text is aligned to the center"
```

XOffset Property

Syntax *Field*.**XOffset**

Data Type Long (read/write)

Description The XOffset property defines the offset, in the horizontal direction, of the field. The offset is relative to the starting position of the field as defined by the FieldPosition property. The offset is measured in twips (1440 twips = 1 inch).

Example The following example illustrates how the XOffset and YOffset properties work.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a current date field to the shape
Set igxField1 = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextCurrentDate, "", ixFieldBelow)
MsgBox "View the diagram"
' Make the maximum width of the field 1 inch
igxField1.MaximumWidth = 1440
MsgBox "Set X offset to 720"
' Set xoffset to 720 twips
igxField1.XOffset = 720
MsgBox "View the change. Now set the Y offset to 720"
' Set yoffset to 720 twips
igxField1.YOffset = 720
MsgBox "End of example"
```

See Also YOffset property

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

YOffset Property

Syntax *Field*.**YOffset**

Data Type Long (read/write)

Description The YOffset property defines the offset, in the vertical direction, of the field. The offset is relative to the starting position of the field as defined by the FieldPosition property. The offset is measured in twips (1440 twips = 1 inch).

Example See the example given for the XOffset property.

See Also XOffset property

{button Field object,JI(`igrafxrf.HLP',`Field_Object')}

Fields Object

The Fields object is a collection of individual Field objects. A Fields collection is only associated with and accessible from a DiagramObject object. Its purpose is to store and provide access to the individual Field objects that are associated with a DiagramObject.

The Fields object provides the following functionality:

- The ability to access any Field objects that have been created for a particular DiagramObject object.
- The ability to determine how many Field objects are in the collection.
- The ability to add a new Field to a DiagramObject object.

Properties, Methods, and Events

All of the properties, methods, and events for the Fields object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent		

Related Topics

Field object

iGrafx API Object Hierarchy

Add Method

Syntax *Fields*.**Add**(*FieldTextType* As IxFieldTextType, *FieldTextData*, [*Position* As IxFieldPosition = ixFieldOtherPosition]) As Field

Description The Add method is used to define new Field objects for a DiagramObject. The method's arguments specify the field type, the data it contains, and its position.

The *FieldTextType* argument specifies the type of data that is displayed or stored in the Field. The IxFieldTextType constant defines the valid values, and are shown in the table below.

Value	Name of Constant
-1	ixFieldTextNone
0	ixFieldTextPageNumber
1	ixFieldTextPageCount
2	ixFieldTextDiagramName
3	ixFieldTextFileName
4	ixFieldTextCurrentDate
5	ixFieldTextCreateDate
6	ixFieldTextSaveDate
7	ixFieldTextCustomData
8	ixFieldTextCustomDataBlock
10	ixFieldTextExpression
12	ixFieldTextShapeNote
13	ixFieldTextShapeNumber

The *FieldTextData* argument specifies the data to display in the field. It's setting depends on the value of the *FieldTextType* argument. Refer to the table below for FieldTextTypes, and the data that must be supplied for each type, using the *FieldTextData* argument. When the *FieldTextData* argument is not used by the *FieldTextType*, you must supply empty double quotes("") for the *FieldTextData* argument.

Value	If you use this *FieldTextType*	Supply this data for *FieldTextData*
-1	IxFieldTextNone	A string to display
0	IxFieldTextPageNumber	Unused. Supply empty quotes "". Displays the DiagramObject's page number.
1	IxFieldTextPageCount	Unused. Supply empty quotes "". Displays the page count of the DiagramObject's document.
2	IxFieldTextDiagramName	Unused. Supply empty quotes "". Displays the name of the DiagramObject's diagram.
3	IxFieldTextFileName	Unused. Supply empty quotes "". Displays the file name of the DiagramObject's document.
4	IxFieldTextCurrentDate	A numeric variant. It controls the time/date format displayed. Use the values from IxDateFormatType.
5	IxFieldTextCreateDate	A numeric variant. It controls the time/date format displayed. Use the values from

		IxDateFormatType.
6	IxFieldTextSaveDate	A numeric variant. It controls the time/date format displayed. Use the values from IxDateFormatType.
7	ixFieldTextCustomData	A string specifying the CustomDataValue's name, or a number specifying it's ID.
8	ixFieldTextCustomDataBlock	Unused. Supply empty quotes "". Displays the DiagramObject's entire custom data block.
10	ixFieldTextExpression	A Visual Basic expression as a string; for example, "ThisShape.DiagramObject.Name".
12	ixFieldTextShapeNote	Unused. Supply empty quotes "". Displays the DiagramObject's shape note, if available.
13	IxFieldTextShapeNumber	Unused. Supply empty quotes "". Displays the DiagramObject's shape number, if available.

The *Position* argument specifies a Field Position. There are a variety of pre-defined field postions defined with the IxFieldPostion enumerated constant. Refer to the Field.FieldPosition property for valid settings.

Example

The following example add three fields to a shape using the Add method: a DiagramName field, a CurrentDate field, and an Expression field.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
Dim igxField2 As Field
Dim igxField3 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a DiagramName field
MsgBox "Click OK to add three fields"
Set igxField1 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextDiagramName, "", ixFieldAbove)
' Add CurrentDate field
Set igxField2 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextCurrentDate, "", ixFieldInsideCenter)
' Add an Expression field
Set igxField3 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextExpression, "Now", ixFieldBelow)
MsgBox "Click OK to continue"
```

{button Fields object,JI(`igrafxrf.HLP',`Fields_Object')}

Item Method

Syntax *Fields*.**Item**(Index) As Field

Description The Item method returns the Field object at the specified *Index* from the Fields collection. The Fields collection contains all of the field objects that have been defined for the current diagram. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Field. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example uses the Item method to iterate through a Shape's Fields collection and change the font color for each Field.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
Dim igxField2 As Field
Dim igxField3 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a DiagramName field
MsgBox "Click OK to add three fields"
Set igxField1 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextDiagramName, "", ixFieldAbove)
' Add CurrentDate field
Set igxField2 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextCurrentDate, "", ixFieldInsideCenter)
' Add an Expression field
Set igxField3 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextExpression, "Now", ixFieldBelow)
With igxShape.DiagramObject.Fields
    For Index = 1 To .Count
        MsgBox "Color the next Item"
        .Item(Index).Font.Color = vbGreen
    Next Index
End With
MsgBox "Click OK to continue"
```

{button Fields object,JI(`igrafxrf.HLP',`Fields_Object')}

FieldText Object

The FieldText object holds the properties and methods associated with a Field in a DiagramObject. Once a Field is added to a DiagramObject, the FieldText object can be used to display the contents of the Field.

Properties, Methods, and Events

All of the properties, methods, and events for the FieldText object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
CustomDataDefinition		
DateFormat		
FormattedValue		
Macro		
NumberFormat		
Parent		
Type		

CustomDataDefinition Property

Syntax *FieldText*.**CustomDataDefinition**

Data Type CustomDataDefinition object (read-only, See Object Properties)

Description The CustomDataDefinition property returns the CustomDataDefinition object that is associated with the specified FieldText object. If the Field does not display custom data, that is, the type is not *ixFieldTextCustomData*, then this property returns an error.

Example The following example creates a shape, a CustomDataDefinition, and a custom data Field on the shape. It then uses the CustomDataDefinition property to access the CustomDataDefinition associated with the Field, and displays it's name.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a custom data definition to the document
ActiveDocument.CustomDataDefinitions.Add _
   "Cost", ixCustomDataFormatCurrencyBase
' Set the shape's CustomDataValue
igxShape.DiagramObject.CustomDataValues.Item _
   ("Cost", ixCustomDataCurrency).Value = 125
' Add a custom data field
MsgBox "Click OK to add a custom data Field"
Set igxField1 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextCustomData, "Cost", ixFieldAbove)
' Display the custom data definition's name
With igxField1.FieldText.CustomDataDefinition
    MsgBox "The custom data definition is called " & .Name
End With
```

See Also CustomDataDefinition object

iGrafx API Object Hierarchy

{button FieldText object,JI(`igrafxrf.HLP',`FieldText_Object')}

DateFormat Property

Syntax *FieldText*.**DateFormat**

Data Type IxDateFormatType enumerated constant (read/write)

Description The DateFormat property specifies a date format to use for the field text, when the Field displays a date. There are 16 different formats available. This property is relevant when the FieldText is of type *IxFieldTextCurrentDate*, *IxFieldTextCreateDate*, or *IxFieldTextCreateDate*.

The IxDateFormatType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixDateFieldCodeDayDateStr
1	ixDateFieldCodeDateStr
2	ixDateFieldCodeDateYear
3	ixDateFieldCodeYearMonthDay
4	ixDateFieldCodeDayMonthYear
5	ixDateFieldCodeMonthDayYearDot
6	ixDateFieldCodeMonthStrDayYear
7	ixDateFieldCodeDayMonthYearStr
8	ixDateFieldCodeMonthYear
9	ixDateFieldCodeMonthYear2
10	ixDateFieldCodeDateTime
11	ixDateFieldCodeDateTimeSec
12	ixDateFieldCodeTime
13	ixDateFieldCodeTimeSec
14	ixDateFieldCodeTime24
15	ixDateFieldCodeTime24Sec

Example The following example steps through each DateFormat type and displays it.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
Set igxField1 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextCurrentDate, "", ixFieldBelow)
For Index = 0 To 15
   igxField1.FieldText.DateFormat = Index
   ' Have a messagebox display the field text
   MsgBox "This is date format" & Str(Index)
Next Index
```

{button FieldText object,JI(`igrafxrf.HLP',`FieldText_Object')}

FormattedValue Property

Syntax *FieldText*.**FormattedValue**

Data Type String (read-only)

Description The FormattedValue property returns the value of the FieldText as a formatted value. This property is relevant only for FieldText objects that display CustomDataValues; that is, the type is *ixFieldTextCustomData*. For instance, if the custom data is a currency data type, the FormattedValue property returns a string complete with dollar sign and two decimal places.

The actual formatting is determined by the CustomDataDefinition associated with the CustomDataValue being displayed. Refer to the CustomDataDefinition object for more information.

Example The following example creates a Field that displays currency using a CustomDataValue object. The currency value is then displayed using the FormattedValue property.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Add a custom data definition to the document
ActiveDocument.CustomDataDefinitions.Add _
   "Cost", ixCustomDataFormatCurrencyBase
' Set the shape's CustomDataValue
igxShape.DiagramObject.CustomDataValues.Item _
   ("Cost", ixCustomDataCurrency).Value = 125
' Add a custom data field
MsgBox "Click OK to add a custom data Field"
Set igxField1 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextCustomData, "Cost", ixFieldAbove)
' Display the custom data definition's name
MsgBox "Formatted value: " & igxField1.FieldText.FormattedValue
```

See Also CustomDataDefinition object

{button FieldText object,JI(`igrafxrf.HLP',`FieldText_Object')}

Macro Property

Syntax *FieldText*.**Macro**

Data Type String (read-only)

Description The Macro property returns the macro expression defined for the field, as a string. This property is relevant only for FieldText objects that display a Visual Basic expression; that is, the type is *IxFieldTextExpression*.

Example The following example creates a shape that has one expression field. The expression field displays the shape's object name. A message is used to display the field's macro.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField As Field
Dim igxFieldText As FieldText
Dim igxCustData As CustomDataValue
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Specify the shape's name
igxShape.DiagramObject.ObjectName = "Shape A"
' Add an expression field to the shape
Set igxField = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextExpression, "ThisShape.DiagramObject.ObjectName")
' Get the field's FieldText object
Set igxFieldText = igxField.FieldText
' Display the FieldText macro
MsgBox "The macro used to display the shape's name is:" & Chr(13) _
    & igxFieldText.Macro
```

See Also Fields.Add method

{button FieldText object,JI(`igrafxrf.HLP',`FieldText_Object')}

NumberFormat Property

Syntax *FieldText*.**NumberFormat**

Data Type NumberFormat object (read-only, See Object Properties)

Description The NumberFormat property returns a NumberFormat object for the specified FieldText object. This property is relevant only for FieldText objects that display numbers; that is, the type is *IxFieldTextShapeNumber*.

Errors If the FieldText object is not of type *ixFieldTextShapeNumber*, attempting to access this object produces a run-time error.

Example The following example creates a shape with a shape number field. It then changes the field's format using the NumberFormat property.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField As Field
Dim igxFieldText As FieldText
Dim igxNumberFormat As NumberFormat
' Add a shape to the diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Specify the shape's name
igxShape.DiagramObject.ObjectName = "Shape A"
' Add a page field to the shape
Set igxField = igxShape.DiagramObject.Fields.Add _
    (ixFieldTextShapeNumber, igxShape)
' Get the field's FieldText object
Set igxFieldText = igxField.FieldText
' Get the field's NumberFormat object
MsgBox "Click OK to change the number format."
Set igxNumberFormat = igxFieldText.NumberFormat
igxNumberFormat.Prefix = "Shape No."
MsgBox "Click OK to continue."
```

See Also NumberFormat object

iGrafx API Object Hierarchy

{button FieldText object,JI(`igrafxrf.HLP',`FieldText_Object')}

Type Property

Syntax *FieldText*.**Type**

Data Type IxFieldTextType enumerated constant (read-only)

Description The Type property returns the “type” of the FieldText object. Use this property to verify that you have the correct field type before accessing objects that are only valid for particular data types.

The IxFieldTextType constant defines the valid values for this property, and are listed in the following table.

Value	Name of Constant
-1	ixFieldTextNone
0	ixFieldTextPageNumber
1	ixFieldTextPageCount
2	ixFieldTextDiagramName
3	ixFieldTextFileName
4	ixFieldTextCurrentDate
5	ixFieldTextCreateDate
6	ixFieldTextSaveDate
7	ixFieldTextCustomData
8	ixFieldTextCustomDataBlock
10	ixFieldTextExpression
12	ixFieldTextShapeNote
13	ixFieldTextShapeNumber

Example The following example sets up a shape with a field in the Main() subroutine, and defines a function called “GetFieldTextType”. The function returns a FieldText' object’s type as a string. The Main( ) subroutine shows one way it can be used.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxShape As Shape
    Dim igxField As Field
    Dim igxFieldText As FieldText
    ' Add a shape to the diagram
    Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    ' Add a page field to the shape
    Set igxField = igxShape.DiagramObject.Fields.Add _
       (ixFieldTextFileName, "", ixFieldAbove)
    ' Get the field's FieldText object
    Set igxFieldText = igxField.FieldText
    ' Get the field's type using our function
    MsgBox "The field is of type: ixFileText" & _
       GetFieldTextType(igxFieldText)
End Sub

' Function returns a string indicating FieldText type
Public Function GetFieldTextType(FieldText As FieldText) _
As String
    Select Case FieldText.Type
```

```
            Case -1
                GetFieldTextType = "None"
            Case 0
                GetFieldTextType = "PageNumber"
            Case 1
                GetFieldTextType = "PageCount"
            Case 2
                GetFieldTextType = "DiagramName"
            Case 3
                GetFieldTextType = "FileName"
            Case 4
                GetFieldTextType = "CurrentDate"
            Case 5
                GetFieldTextType = "CreateDate"
            Case 6
                GetFieldTextType = "SaveDate"
            Case 7
                GetFieldTextType = "CustomData"
            Case 8
                GetFieldTextType = "CustomDataBlock"
            Case 10
                GetFieldTextType = "Expression"
            Case 12
                GetFieldTextType = "ShapeNote"
            Case 13
                GetFieldTextType = "ShapeNumber"
        End Select
    End Function
```

{button FieldText object,JI(`igrafxrf.HLP',`FieldText_Object')}

FieldTexts Object

The FieldTexts object is the collection of FieldText objects that are contained in a particular TextRange object. TextRange objects can be found under the following types of objects that use text in some way:

- TextGraphicObject
- Department
- HeaderFooter
- Note
- TextBlock
- ChildTextBlock
- Paragraph

With the TextRange.InsertFieldText method, you can insert a FieldText object into any of the aforementioned objects. For instance, you could display a CurrentDate field within a Department's name area by inserting a date FieldText into the Department object's TextRange property.

You can insert more than one FieldText object into any given text range. Once inserted, each FieldText object can be accessed using the FieldTexts collection for that specific text range. Use the Item method to access individual FieldText objects in the FieldTexts collection.

The FieldTexts object provides the following functionality:

- The ability to access any FieldText objects in the collection.
- The ability to determine how many FieldText objects are in the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the FieldTexts object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
Parent		

Item Method

Syntax *FieldTexts*.**Item**(Index) As FieldText

Description The Item method returns the FieldText object at the specified *Index* from the FieldTexts collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type FieldText. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example inserts a file name field and a date field into the TextRange of a TextGraphic object. It then uses the Item method to access the date field and change it's format.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxField1 As Field
Dim igxField2 As Field
Dim igxRange As TextRange
Dim igxText As TextGraphicObject
' Create a shape in the active diagram
Set igxShape = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2)
' Create a text graphic object
Set igxText = ActiveDiagram.DiagramObjects.AddTextObject _
   (1440 * 5, 1440)
' Get the text graphic's TextRange
Set igxRange = igxText.TextRange
' Add a file name field to the shape
Set igxField1 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextFileName, "", ixFieldBelow)
' Add a date field to the shape
Set igxField2 = igxShape.DiagramObject.Fields.Add _
   (ixFieldTextCurrentDate, "", ixFieldAbove)
' Insert 2 fields in the TextRange
igxRange.Text = Chr(13)
igxRange.InsertFieldText 2, ixFieldTextCurrentDate, ""
igxRange.InsertFieldText 1, ixFieldTextFileName, ""
' Change the date format of the date field
MsgBox "Click OK to change the date format on the text graphic."
igxRange.FieldTexts.Item(1).DateFormat = ixDateTextDateTime
MsgBox "Click OK to continue"
```

{button FieldTexts object,JI(`igrafxrf.HLP',`FieldTexts_Object')}

CustomDataDefinition Object

The CustomDataDefinition object defines a data field to be associated with DiagramObject objects. For instance, if you are developing an organization diagram where each shape represents a person, you might have custom data “definitions” such as Name, Title, Location, Department Number, and Telephone Number. You can think of this object as being the description of a data field, similar to defining a variable in a programming language.

When you add a CustomDataDefinition to a document, all the DiagramObjects in the document inherit a CustomDataValue object. The new CustomDataValue can be set to a different value for each DiagramObject object. The data in each CustomDataValue is a member of a data set accumulated by the CustomDataDefinition. The accumulated data can then be displayed using a variety of statistical methods and display formats.

A document can contain more than one custom data definition; therefore, the CustomDataDefinitions object is a collection of individual CustomDataDefinition objects. The collection is attached at the Document level, and so all individual data definitions apply to every diagram and every DiagramObject in the document.

A custom data definition has associated data values. These values are represented in the API by the following objects:

- CustomDataValue: an individual value
- CustomDataValues: the collection of values associated with a particular CustomDataDefinition object

Refer to the iGrafx API Object Hierarchy for information about the object relationships.

Properties, Methods, and Events

All of the properties, methods, and events for the CustomDataDefinition object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Accumulation	Delete	
AccumulationMethod		
Application		
Format		
Hidden		
Name		
Parent		
Type		

Accumulation Property

Syntax *CustomDataDefinition*.**Accumulation**

Data Type Double (read-only)

Description The Accumulation property returns the accumulated value of a CustomDataDefinition object. All the CustomDataValue values associated with the CustomDataDefinition are accumulated as a data set, and this property returns the accumulated value.

Exampe The following example sets up a flowchart with two shapes. A CustomDataDefinition object is added to the document called "MyCost". In each shape, the "MyCost" CustomDataValue is set to a different cost. The MyCost data accumulates in the CustomDataDefinition object. The accumulated data is then displayed using each of the AccumulationMethod types.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxDataDef As CustomDataDefinition
Dim igxValue As CustomDataValue
' Add shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Make Shape 1 a start point
igxShape1.StartPointName = "Start"
' Add text to each shape
igxShape1.Text = "$10"
igxShape2.Text = "$5"
' Add a connector between the shapes
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
    "MyCost", ixCustomDataFormatCurrencyBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Set the value of each shape's CustomDataValue
igxShape1.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataCurrency).Value = 10
igxShape2.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataCurrency).Value = 5
' Display the accumulated value in various ways
igxDataDef.AccumulationMethod = ixSum
MsgBox "Sum: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMean
MsgBox "Mean: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMedian
MsgBox "Median: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMin
MsgBox "Min: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMax
MsgBox "Max: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixRange
MsgBox "Range: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixObjectCount
MsgBox "ObjectCount: " & igxDataDef.Accumulation
```

```
igxDataDef.AccumulationMethod = ixFilledCount
MsgBox "FilledCount: " & igxDataDef.Accumulation
```

See Also AccumulationMethod property

{button CustomDataDefinition object,JI(`igrafxrf.HLP',`CustomDataDefinition_Object')}

AccumulationMethod Property

Syntax *CustomDataDefinition*.**AccumulationMethod**

Data Type IxAccumulationMethod enumerated constant (read/write)

Description The AccumulationMethod property specifies which type of statistical calculation to use when displaying accumulated data from a CustomDataDefinition. This setting also determines how accumulated data is displayed in a Legend.

The IxAccumulationMethod constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixNoMethod
1	ixSum
2	ixMean
3	ixMedian
4	ixMin
5	ixMax
6	ixRange
7	ixObjectCount
8	ixFilledCount

Example Refer the example for the Accumulation property.

{button CustomDataDefinition object,JI(`igrafxrf.HLP',`CustomDataDefinition_Object')}

Format Property

Syntax *CustomDataDefinition*.**Format**

Data Type IxFieldFormat enumerated constant (read/write)

Description The Format property specifies the data format used by the CustomDataDefinition object and it's corresponding CustomDataValue objects. For instance, if the CustomDataDefinition is of type *IxCustomDataDuration*, you may want to express time values as "hh:mm:ss". In this case, you would set the Format property to *IxFieldFormatHMS*.

Note The Format property should be used before setting any custom data values. If you change the Format property, the current dataset stored in the CustomDataDefinition will be of the wrong format and, therefore, unusable.

The IxFieldFormat constant defines the valid values for this property, which are listed in the following table.

.

Value	Name of Constant
0	ixFieldFormatText
1	ixFieldFormatWDotWeeks
2	ixFieldFormatWeeks
3	ixFieldFormatDDotDays
4	ixFieldFormatDays
5	ixFieldFormatHDotHours
6	ixFieldFormatHRSDotHours
7	ixFieldFormatHours
8	ixFieldFormatMDotMinutes
9	ixFieldFormatMinMinutes
10	ixFieldFormatMinutes
11	ixFieldFormatSDotSeconds
12	ixFieldFormatSecsSeconds
13	ixFieldFormatSeconds
14	ixFieldFormatTMU
15	ixFieldFormatHM
16	ixFieldFormatMS
17	ixFieldFormatHMS
18	ixFieldFormatShortDate
19	ixFieldFormatSpecialShortDate
20	ixFieldFormatLongDate
21	ixFieldFormatShortMonthYear
22	ixFieldFormatLongMonthYear
23	ixFieldFormatStdCurrency
24	ixFieldFormatStdCurrencyCommas
25	ixFieldFormatNoPennies
26	ixFieldFormatNoPenniesCommas
27	ixFieldFormatWholePercent
28	ixFieldFormatDecimalPercent
29	ixFieldFormatInteger
30	ixFieldFormatTwoPrecision

31	ixFieldFormatFourPrecision
32	ixFieldFormatIntegerCommas
33	ixFieldFormatTwoPrecisionCommas
34	ixFieldFormatFourPrecisionCommas

Example The following example sets up a CustomDataValue that stores a time. It then displays the time using the FormattedValue property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxValue1 As CustomDataValue
' Add shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a legend
ActiveDiagram.DiagramObjects.AddLegend 3000, 4000
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add "MyTime", ixCustomDataFormatTimeBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Get the shape's CustomDataValue object
Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataDuration)
' Set the format for the time data
igxDataDef.Format = ixFieldFormatHMS
' Set the value of the shape's CustomDataValue
igxShape1.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataDuration).Value = "12:05:34"
' Report the time components of value
MsgBox "Formatted duration: " & igxValue1.FormattedValue
```

See Also Type property

{button CustomDataDefinition object,JI(`igrafxrf.HLP',`CustomDataDefinition_Object')}

Hidden Property

Syntax *CustomDataDefinition*.**Hidden**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Hidden property specifies whether a CustomDataDefinition object's value is displayed in a Legend. If set to True, the value appears in a Legend, if one exists in the diagram. If set to False, the value does not appear in a Legend.

Example The following example sets up a flowchart with two shapes. A CustomDataDefinition is added to the document called "MyCost". In each shape, the "MyCost" CustomDataValue is set to a different cost. The MyCost data accumulates in the CustomDataDefinition. The accumulated data is displayed in a Legend. The Hidden property is then used to hide and unhide the display in the Legend.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxDataDef As CustomDataDefinition
Dim igxValue As CustomDataValue
' Add shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Make Shape 1 a start point
igxShape1.StartPointName = "Start"
' Add text to each shape
igxShape1.Text = "$5.00"
igxShape2.Text = "$10.00"
' Add a connector between the shapes
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add a legend
ActiveDiagram.DiagramObjects.AddLegend 3000, 4000
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
    "MyCost",ixCustomDataFormatCurrencyBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Set the value of each shape's CustomDataValue
igxShape1.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataCurrency).Value = 5
igxShape2.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataCurrency).Value = 10
' Set the accumulation method
igxDataDef.AccumulationMethod = ixSum
' Hide and Unhide the value in the Legend
MsgBox "Click OK to hide the CustomDataDefinition in the Legend"
igxDataDef.Hidden = True
MsgBox "Click OK to restore the CustomDataDefinition in the Legend"
igxDataDef.Hidden = False
MsgBox "Click OK to continue."
```

{button CustomDataDefinition

object,JI(`igrafxrf.HLP',`CustomDataDefinition_Object')}

Type Property

Syntax *CustomDataDefinition*.**Type**

Data Type IxCustomDataType enumerated constant (read/write)

Description The Type property specifies the data type of the CustomDataDefinition object, and it's associated CustomDataValue objects.

The IxCustomDataType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixCustomDataText
1	ixCustomDataDuration
2	ixCustomDataDate
3	ixCustomDataCurrency
4	ixCustomDataPercent
5	ixCustomDataNumber

Note The Type property is related to the Format property. If you change the Type property of a CustomDataDefinition, the default Format for that Type is automatically applied, until changed by the program.

Example The following example sets up a CustomDataDefinition in the document. It then uses the Type property to determine the data type of the CustomDataDefinition object.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxDataDef As CustomDataDefinition
Dim igxValue As CustomDataValue
' Add shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Make Shape 1 a start point
igxShape1.StartPointName = "Start"
' Add text to each shape
igxShape1.Text = "$5.00"
igxShape2.Text = "$10.00"
' Add a connector between the shapes
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add a legend
ActiveDiagram.DiagramObjects.AddLegend 3000, 4000
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
    "MyCost", ixCustomDataFormatCurrencyBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Set the value of each shape's CustomDataValue
igxShape1.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataCurrency).Value = 5
igxShape2.DiagramObject.CustomDataValues.Item _
```

```
    (1, ixCustomDataCurrency).Value = 10
' Display the accumulated value in various ways
igxDataDef.AccumulationMethod = ixSum
' Determine the data type
Select Case igxDataDef.Type
    Case 0
        sString = "Text"
    Case 1
        sString = "Duration"
    Case 2
        sString = "Date"
    Case 3
        sString = "Currency"
    Case 4
        sString = "Percent"
    Case 5
        sString = "Number"
End Select
MsgBox "The CustomDataDefinition is of type " & sString
```

See Also Format property

{button CustomDataDefinition object,JI(`igrafxrf.HLP',`CustomDataDefinition_Object')}

CustomDataDefinitions Object

The CustomDataDefinitions object is a collection of individual CustomDataDefinition objects. A CustomDataDefinitions collection is only associated with and accessible from a Document object. Its purpose is to store and provide access to the individual CustomDataDefinition objects that have been created for a document.

The CustomDataDefinitions object provides the following functionality:

- The ability to access any CustomDataDefinition objects that have been created for a particular Document object.
- The ability to determine how many CustomDataDefinition objects are in the collection.
- The ability to add a new CustomDataDefinition object to a document.

Properties, Methods, and Events

All of the properties, methods, and events for the CustomDataDefinitions object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent		

Related Topics

CustomDataDefinition object

iGrafx API Object Hierarchy

Add Method

Syntax *CustomDataDefinitions*.**Add**(*FieldName* As String, *FieldType* As IxCustomDataFormat) As CustomDataDefinition

Description The Add method adds a CustomDataDefinition object to the CustomDataDefinitions collection of a document. The result of the method is a CustomDataDefinition object.

The *FieldName* argument specifies the name of the CustomDataDefinition as it appears in a Legend. The value can be any string you choose.

The *FieldType* argument specifies the type of values that the CustomDataDefinition is accumulating. The IxCustomDataFormat constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixCustomDataFormatTextBase
1	ixCustomDataFormatTimeBase
2	ixCustomDataFormatDateBase
3	ixCustomDataFormatCurrencyBase
4	ixCustomDataFormatPercentBase
5	ixCustomDataFormatGeneralBase

Example The following example sets up a flowchart with two shapes. Using the Add method a CustomDataDefinition is added to the document called "MyCost". In each shape, the "MyCost" CustomDataValue is set to a different cost. The MyCost data accumulates in the CustomDataDefinition. The accumulated data is then displayed using each of the AccumulationMethod types.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxDataDef As CustomDataDefinition
Dim igxValue As CustomDataValue
' Add shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Make Shape 1 a start point
igxShape1.StartPointName = "Start"
' Add text to each shape
igxShape1.Text = "$10"
igxShape2.Text = "$5"
' Add a connector between the shapes
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
    "MyCost", ixCustomDataFormatCurrencyBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Set the value of each shape's CustomDataValue
igxShape1.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataCurrency).Value = 10
igxShape2.DiagramObject.CustomDataValues.Item _
```

```
    (1, ixCustomDataCurrency).Value = 5
' Display the accumulated value in various ways
igxDataDef.AccumulationMethod = ixSum
MsgBox "Sum: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMean
MsgBox "Mean: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMedian
MsgBox "Median: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMin
MsgBox "Min: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMax
MsgBox "Max: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixRange
MsgBox "Range: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixObjectCount
MsgBox "ObjectCount: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixFilledCount
MsgBox "FilledCount: " & igxDataDef.Accumulation
```

{button CustomDataDefinitions object,JI(`igrafxrf.HLP',`CustomDataDefinitions_Object')}

Item Method

Syntax *CustomDataDefinitions*.**Item**

Description The Item method returns the CustomDataDefinition object at the specified *Index* from the CustomDataDefinitions collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type CustomDataDefinition. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example sets up a flowchart with two shapes. A CustomDataDefinition is added to the document called "MyCost". The Item method is used to set a variable to the CustomDataDefinition object. In each shape, the "MyCost" CustomDataValue is set to a different cost. The MyCost data accumulates in the CustomDataDefinition. The accumulated data is then displayed using each of the AccumulationMethod types.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxDataDef As CustomDataDefinition
Dim igxValue As CustomDataValue
' Add shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Make Shape 1 a start point
igxShape1.StartPointName = "Start"
' Add text to each shape
igxShape1.Text = "$10"
igxShape2.Text = "$5"
' Add a connector between the shapes
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
    "MyCost", ixCustomDataFormatCurrencyBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Set the value of each shape's CustomDataValue
igxShape1.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataCurrency).Value = 10
igxShape2.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataCurrency).Value = 5
' Display the accumulated value in various ways
igxDataDef.AccumulationMethod = ixSum
MsgBox "Sum: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMean
MsgBox "Mean: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMedian
MsgBox "Median: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMin
MsgBox "Min: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixMax
MsgBox "Max: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixRange
```

```
MsgBox "Range: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixObjectCount
MsgBox "ObjectCount: " & igxDataDef.Accumulation
igxDataDef.AccumulationMethod = ixFilledCount
MsgBox "FilledCount: " & igxDataDef.Accumulation
```

{button CustomDataDefinitions object,JI(`igrafxrf.HLP',`CustomDataDefinitions_Object')}

CustomDataValue Object

The CustomDataValue object stores a value for a CustomDataDefinition. When a CustomDataDefinition of a particular type is added to a diagram, every DiagramObject object in the document inherits a CustomDataValue of that type. Each individual CustomDataValue can be given a value, and those values become part of a data set. The data set can be evaluated using several different statistical calulation methods using the CustomDataDefinition.Accumulation property.

CustomDataValue objects have a Type, such as Number, Text, Currency, Date, or Duration. The type of the data value is specified by the CustomDataDefinition of which the data value is a member.

Properties, Methods, and Events

All of the properties, methods, and events for the CustomDataValue object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Compare	
CustomDataDefinition	Empty	
Day		
FormattedValue		
Hours		
IsEmpty		
Minutes		
Month		
Name		
Parent		
Seconds		
TMUs		
Type		
Value		
Weeks		
Year		

Compare Method

Syntax *CustomDataValue*.**Compare**(*Value*) As Integer

Description The Compare method determines if some value is lower, equal to, or higher than the specified CustomDataValue object. The method returns an integer which indicates the result of the comparison. This is useful for comparing dates, for instance. The *Value* argument is the value to compare.

The following table shows the values returned by the Compare method.

Result	Return Value
CustomDataValue is lower than the argument value	-1
CustomDataValue is equal to the argument value	0
CustomDataValue is higher than the argument value	1

Example The following example creates a shape with a CustomDataValue stored as a time. The Compare method is then used to determine if a supplied time comes before or after the CustomDataValue's time.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxValue1 As CustomDataValue
Dim TimeValue As Variant
' Add shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
   "Time", ixCustomDataFormatTimeBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
igxDataDef.Format = ixFieldFormatHMS
' Get the shape's CustomDataValue object
Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
   ("Time", ixCustomDataDuration)
' Set the value of the shape's CustomDataValue
igxValue1.Value = "930"
TimeValue = "800"
Select Case igxValue1.Compare(TimeValue)
    Case 1
        MsgBox "The supplied time " & TimeValue & ", comes before 9:30."
    Case 0
        MsgBox "The supplied time is 9:30."
    Case -1
        MsgBox "The supplied time " & TimeValue & ", comes after 9:30."
End Select
```

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

CustomDataDefinition Property

Syntax *CustomDataValue*.**CustomDataDefinition**

Data Type CustomDataDefinition object (read-only, See Object Properties)

Description The CustomDataDefinition property returns the CustomDataDefinition object with which the CustomDataValue is associated—the CustomDataDefinition to which the CustomDataValue sends accumulation data.

Example The following example uses the CustomDataValue object's CustomDataDefinition property to delete it's own CustomDataDefinition.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxValue1 As CustomDataValue
' Add shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
   "MyDate", ixCustomDataFormatTimeBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
igxDataDef.Format = ixFieldFormatHMS
' Get the shape's CustomDataValue object
Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataDuration)
' Set the value of the shape's CustomDataValue
igxValue1.Value = "00:30:00"
' Display the value
MsgBox "The value is " & igxValue1.FormattedValue & _
Chr(13) & "Click OK to delete the CustomDataDefinition."
' Delete the CustomDataDefinition
igxValue1.CustomDataDefinition.Delete
MsgBox "CustomDataDefinition deleted."
```

See Also CustomDataDefinition object

iGrafx API Object Hierarchy

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

Day Property

Syntax *CustomDataValue*.**Day**

Data Type Integer (read/write)

Description The Day property is valid when the “type” for the data value’s CustomDataDefinition is one of the following:

- ixCustomDataDate
- ixCustomDataDuration

If the type is ixCustomDataDate, the Day property returns the day component of the date. For instance, if the CustomDataValue is storing "March/24/1999", the Day property would return the number 24.

If the type is ixCustomDataDuration (it stores a time duration), the Day property returns a day of the month based on the amount of time elapsed since midnight January 1. For instance, if the CustomDataValue is storing "912:00:00" (Hours:Minutes:Seconds format), the Day property returns the number 7, because 912 hours since midnight January 1 would be February 7.

Example The following example creates a CustomDataDefinition to store a date. A CustomDataValue object's value is set to a date, and the Month, Day, and Year properties are displayed, as well as the number of weeks elapsed in the month.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxValue1 As CustomDataValue
' Add shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a legend
ActiveDiagram.DiagramObjects.AddLegend 3000, 4000
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
   "MyDate", ixCustomDataFormatDateBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Get the shape's CustomDataValue object
Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataDate)
' Set the value of the shape's CustomDataValue
igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataDate).Value = "March/4/1999"
' Display the Day, Month, Year, and Weeks properties of the date
MsgBox "The stored date is the " & igxValue1.Day & "th day of the " _
   & igxValue1.Month & "rd month, in the year " & igxValue1.Year _
   & Chr(13) & gxValue1.Weeks & " weeks have elapsed in the month."
```

See Also Month property

Weeks property

Year property

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

Empty Method

Syntax *CustomDataValue*.**Empty**

Description The Empty method clears the value that is stored in the specified CustomDataValue object.

Example The following example shows a diagram Legend before and after using the Empty method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxValue1 As CustomDataValue
' Add shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a legend
ActiveDiagram.DiagramObjects.AddLegend 3000, 4000
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
   "MyTime", ixCustomDataFormatTimeBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Get the shape's CustomDataValue object
Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataDuration)
' Set the value of the shape's CustomDataValue
igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataDuration).Value = 912
' Empty the value
MsgBox "Click OK to empty the CustomDataValue"
igxValue1.Empty
MsgBox "Click OK to continue."
```

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

FormattedValue Property

Syntax *CustomDataValue*.**FormattedValue**

Data Type String (read-only)

Description The FormattedValue property returns the a formatted string value of the CustomDataValue. The formatting of the string depends on the Format type specified by the CustomDataDefinition.Format property. Note that the actual value stored and the formatted value that gets displayed can be different. For example, dates can be formatted in numerous ways while the actual value is not affected.

Example The following example sets up a CustomDataValue that stores a time. It then displays the time using the FormattedValue property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxValue1 As CustomDataValue
' Add shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a legend
ActiveDiagram.DiagramObjects.AddLegend 3000, 4000
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
   "MyTime", ixCustomDataFormatTimeBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Get the shape's CustomDataValue object
Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataDuration)
' Set the value of the shape's CustomDataValue
igxDataDef.Format = ixFieldFormatHMS
igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataDuration).Value = "12:05:34"
' Display the time value, formatted
MsgBox "Formatted duration: " & igxValue1.FormattedValue
```

See Also CustomDataDefinition.Format property

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

Hours Property

Syntax *CustomDataValue*.**Hours**

Data Type Double (read/write)

Description The Hours property specifies a duration of time in units of hours. For instance, if the data value is "00:120:00", the Hours property returns the number 2, because 120 minutes is 2 hours. The property is only valid when the data type of the CustomDataDefinition object is ixCustomDataDuration.

Example The following example creates a CustomDateValue that stores a duration. The value of the CustomDataValue is then displayed, converted in various ways using the Hours, Minutes, Seconds, TMUs, Day, and Month properties.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxValue1 As CustomDataValue
' Add shapes
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a legend
ActiveDiagram.DiagramObjects.AddLegend 3000, 4000
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
    "MyTime", ixCustomDataFormatTimeBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Get the shape's CustomDataValue object
Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataDuration)
' Set the value of the shape's CustomDataValue
igxDataDef.Format = ixFieldFormatHMS
igxShape1.DiagramObject.CustomDataValues.Item _
    (1, ixCustomDataDuration).Value = "01:24:00"
' Display the duration value in various ways
MsgBox igxValue1.FormattedValue & " can be expressed as: " _
    & Chr(13) & Chr(13) & _
    Round(igxValue1.Hours, 1) & "  hours" & Chr(13) & _
    Round(igxValue1.Minutes, 1) & "  minutes" & Chr(13) & _
    Round(igxValue1.Seconds, 0) & "  seconds" & Chr(13) & _
    Round(igxValue1.TMUs, 5) & "  TMUs" & Chr(13) & _
    igxValue1.Day & "  days into the month" & Chr(13) & _
    igxValue1.Month & "  months into the year"
```

See Also Minutes property

Seconds property

TMUs property

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

IsEmpty Property

Syntax *CustomDataValue*.**IsEmpty**[= {**True | False**}]

Data Type Boolean (read-only)

Description The IsEmpty property indicates whether the specified CustomDataValue is empty.

Example The following example sets up a CustomDataValue object. It fills it with a value, and then empties it. Then, using the IsEmpty property, a check is performed to determine whether the CustomDataValue object is empty.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxValue1 As CustomDataValue
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add a legend
ActiveDiagram.DiagramObjects.AddLegend 3000, 4000
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add _
   "MyTime", ixCustomDataFormatTimeBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Get the shape's CustomDataValue object
Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataDuration)
' Set the value of the shape's CustomDataValue
igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataDuration).Value = 912
' Empty the value
MsgBox "Click OK to empty the CustomDataValue"
igxValue1.Empty
If igxValue1.IsEmpty Then
   MsgBox "The CustomDataValue is empty."
End If
```

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

Minutes Property

Syntax *CustomDataValue*.**Minutes**

Data Type Double (read/write)

Description The Minutes property specifies a duration of time in units of minutes. For instance, if the data value is "02:00:00", the Minutes property returns the number 120, because 2 hours is 120 minutes. The property is only valid when the data type of the CustomDataDefinition object is ixCustomDataDuration.

Example Refer to the example for the Hours property.

See Also Hours property

Seconds property

TMUs property

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

Month Property

Syntax *CustomDataValue*.**Month**

Data Type Integer (read/write)

Description The Month property is valid when the “type” for the data value’s CustomDataDefinition is one of the following:

- ixCustomDataDate
- ixCustomDataDuration

If the type is ixCustomDataDate, the Month property returns the month component of the date. For instance, if the CustomDataValue is storing "March/24/1999", the Month property would return the number 3.

If the type is ixCustomDataDuration (it stores a time duration), the Month property returns a month of the year, based on the amount of time elapsed since midnight January 1. For instance, if the CustomDataValue is storing "912:00:00" (Hours:Minutes:Seconds format), the Month property returns the number 2, because 912 hours since midnight January 1 would be February 7.

Example Refer to the example for the Day property.

See Also Day property

Weeks property

Year property

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

Seconds Property

Syntax *CustomDataValue*.**Seconds**

Data Type Double (read/write)

Description The Seconds property specifies a duration of time in units of seconds. For instance, if the data value is "02:00:00", the Seconds property returns the number 216000, because 2 hours is 216000 seconds. The property is only valid when the data type of the CustomDataDefinition object is ixCustomDataDuration.

Example Refer to the example for the Hours property.

See Also Hours property

Minutes property

TMUs property

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

TMUs Property

Syntax *CustomDataValue*.**TMUs**

Data Type Double (read/write)

Description The TMUs property specifies a duration of time in “Time Measurement Units”, or TMUs. For instance, if the data value is "01:00:00", the TMUs property returns the number 100000, because there are 100,000 TMUs in one hour. The property is only valid when the data type of the CustomDataDefinition object is ixCustomDataDuration.

Example Refer to the example for the Hours property.

See Also Hours property

Minutes property

Seconds property

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

Value Property

Syntax *CustomDataValue*.**Value**

Data Type Variant (read/write)

Description The Value property specifies the actual value that the CustomDataValue object stores internally. The value it returns may be different from the value returned by the FormattedValue property. For instance, a duration might be entered as "00:30:00", but the Value property returns 0.49999 (hours.)

Note In the iGrafx Professional API, duration math is based on 1/100,000 of an hour, so values that use minutes and seconds may return irrational decimal values (such as 0.4999999… instead of 0.5) when using the Value property. Use Visual Basic's Round( ) function to round off numbers.

Example The following example sets up a CustomDataValue whose type is a duration, and then displays it's Value property and it's FormattedValue property.

```
Private Sub Main()
   ' Dimension the variables
   Dim igxShape1 As Shape
   Dim igxDataDef As CustomDataDefinition
   Dim igxValue1 As CustomDataValue
   ' Add shapes
   Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
   ' Add a CustomDataDefinition to the document
   ThisDocument.CustomDataDefinitions.Add _
       "MyDate", ixCustomDataFormatDateBase
   Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
   igxDataDef.Format = ixFieldFormatHMS
   ' Get the shape's CustomDataValue object
   Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
       (1, ixCustomDataDuration)
   ' Set the value of the shape's CustomDataValue
   igxValue1.Value = "01:12:45"
   ' Display the type using our CheckType Function
   MsgBox "Formatted Value: " & igxValue1.FormattedValue & _
       Chr(13) & Chr(13) & "Value: " & igxValue1.Value
End Sub
```

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

Weeks Property

Syntax *CustomDataValue*.**Weeks**

Data Type Double (read/write)

Description The Weeks property is valid when the “type” for the data value’s CustomDataDefinition is one of the following:

- ixCustomDataDate
- ixCustomDataDuration

If the type is ixCustomDataDate, the Weeks property returns the number of weeks that have elapsed in the month. For instance, if the CustomDataValue is storing "March/24/1999", the Month property would return the number 3, because 3 weeks have elapsed in that month as of the 24th.

If the type is ixCustomDataDuration (it stores a time duration), the Weeks property returns the number of weeks elapsed in the month, based on the amount of time elapsed since midnight January 1. For instance, if the CustomDataValue is storing "936:00:00", the Weeks property returns the number 1, because 936 hours since midnight January 1 is February 8, and 1 week has elapsed in February as of February 8. (Note that Duration values are always stored internally as hours.) The Weeks property always returns a value from 0 to 4.

Example Refer to the example for the Day property.

See Also Day property

Month property

Year property

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

Year Property

Syntax *CustomDataValue*.**Year**

Data Type Integer (read/write)

Description The Year property is valid when the “type” for the data value’s CustomDataDefinition is one of the following:

- ixCustomDataDate
- ixCustomDataDuration

If the type is ixCustomDataDate, the Year property returns the year component of the date. For instance, if the CustomDataValue is storing "March/24/1999", the Month property would return the number 1999.

If the type is ixCustomDataDuration (it stores a time duration), the Year property returns a year number, based on the amount of time elapsed since midnight January 1, 1900. For instance, if the CustomDataValue is storing the number 9720, the Year property returns the number 1901, because 9720 hours since midnight January 1, 1900 would fall in the year 1901. (Note that Duration values are always stored internally as hours.)

Example Refer to the example for the Day property.

See Also Day property

Month property

Weeks property

{button CustomDataValue object,JI(`igrafxrf.HLP',`CustomDataValue_Object')}

CustomDataValues Object

The CustomDataValues object is a collection of individual CustomDataValue objects. A CustomDataValues collection is associated with and accessible from the following objects:

- DiagramObject object
- Entity object
- Path object

Its purpose is to store and provide access to the individual CustomDataValue objects.

The CustomDataValues object provides the following functionality:

- The ability to access any CustomDataValue objects in the collection.
- The ability to determine how many CustomDataValue objects are in the collection.
- The ability to update all of the CustomDataValue objects in the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the CustomDataValues object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count	UpdateAll	
Parent		

Item Method

Syntax *CustomDataValues*.**Item**(*Index*, [*FieldType* As IxCustomDataType])

Description The Item method returns the CustomDataValue object at the specified *Index* from the CustomDataValues collection.

The *Index* can be a string or an integer. If it's a string, the string specifies the name of the CustomDataValue. If it's an integer, the index number specifies the position of the object in the collection. Use the Count property to determine valid values for the *Index* number. The result of the method must be assigned to a variable of type CustomDataValue. An error is returned if the index is invalid.

The optional *FieldType* argument specifies the field type of the CustomDataValue object. If specifying *Index* as a name string, and there are multiple CustomDataValues with the same name, the FieldType argument specifies which version of the CustomDataValue to return.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example creates two CustomDataDefinitions in the document, and therefore two CustomDataValues for the shape. The Item method is used to assign each CustomDataValue to a variable.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxDataDef1 As CustomDataDefinition
Dim igxDataDef2 As CustomDataDefinition
Dim igxValue1 As CustomDataValue
Dim igxValue2 As CustomDataValue
' Add a shape
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Add two CustomDataDefinitions to the document
ThisDocument.CustomDataDefinitions.Add _
   "Activity", ixCustomDataFormatTextBase
ThisDocument.CustomDataDefinitions.Add _
   "Hours", ixCustomDataFormatTimeBase
' Get the two CustomDataDefinition objects
Set igxDataDef1 = ThisDocument.CustomDataDefinitions.Item(1)
Set igxDataDef2 = ThisDocument.CustomDataDefinitions.Item(2)
' Set the CustomDataDefinition formats
igxDataDef1.Format = ixFieldFormatText
igxDataDef2.Format = ixFieldFormatHours
' Get the shape's CustomDataValue objects
Set igxValue1 = igxShape1.DiagramObject.CustomDataValues.Item _
   (1, ixCustomDataText)
Set igxValue2 = igxShape1.DiagramObject.CustomDataValues.Item _
   (2, ixCustomDataDuration)
' Set the values of the shape's CustomDataValues
igxValue1.Value = "Production"
igxValue2.Value = 8
' Display the values
MsgBox "Activity: " & igxValue1.Value & "  Hours: " & igxValue2.Value
```

{button CustomDataValues object,JI(`igrafxrf.HLP',`CustomDataValues_Object')}

UpdateAll Method

Syntax *CustomDataValues*.**UpdateAll**

Description The UpdateAll method updates all CustomDataValue objects that are accumulated from diagram links. Any links involved must have their AccumulateData property set to True. If a CustomDataValue object in a linked diagram changes, use the UpdateAll method to update the accumulated data in the current diagram.

Example The following example sets up two diagrams with shapes, and adds a CustomDataDefinition to the document. Diagram A has a link to Diagram B. A CustomDataValue is changed on Diagram B. Then the UpdateAll method is used to reflect the change in the Legend in Diagram A.

```
' Dimension the variables
Dim igxDiagramA As Diagram
Dim igxDiagramB As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxLink As Link
' Create two diagrams
Set igxDiagramA = ActiveDocument.Diagrams.Add("Diagram A")
Set igxDiagramB = ActiveDocument.Diagrams.Add("Diagram B")
' Add shapes to Diagram A
Set igxShape1 = igxDiagramA.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = igxDiagramA.DiagramObjects.AddShape(1440 * 3, 1440)
igxDiagramA.DiagramObjects.AddLegend 3000, 3000
' Add shapes to Diagram B
Set igxShape3 = igxDiagramB.DiagramObjects.AddShape(1440, 1440)
Set igxShape4 = igxDiagramB.DiagramObjects.AddShape(1440 * 3, 1440)
igxShape3.StartPointName = "StartB"
' Add a CustomDataDefinition to the document
ActiveDocument.CustomDataDefinitions.Add _
    "Cost", ixCustomDataFormatCurrencyBase
' Set the Shapes' CustomDataValues
igxShape1.DiagramObject.CustomDataValues.Item(1).Value = 10
igxShape2.DiagramObject.CustomDataValues.Item(1).Value = 10
igxShape3.DiagramObject.CustomDataValues.Item(1).Value = 10
igxShape4.DiagramObject.CustomDataValues.Item(1).Value = 10
' Add a link from Diagram A to Diagram B
Set igxLink = igxShape2.Links.AddDiagramLink("Diagram B")
' Have the link accumulate data
igxLink.AccumulateData = True
' Activate Diagram A so we can see it
igxDiagramA.ActivateDiagram
' Change a CustomDataValue on Diagram B
MsgBox "Click to change a CustomDataValue on Diagram B"
igxShape3.DiagramObject.CustomDataValues.Item(1).Value = 20
' Update the accumulation data
igxShape2.DiagramObject.CustomDataValues.UpdateAll
MsgBox "Click OK to continue"
```

{button CustomDataValues object,JI(`igrafxrf.HLP',`CustomDataValues_Object')}

Group Object

The Group object represents a “group” of objects in a diagram. Group objects are created by ObjectRange objects using the ObjectRange.Group method. The result is a Group object, and a group in the iGrafx Professional user interface. The grouped shapes then can be manipulated (moved, rotated, etc.) together from the user interface.

The grouping of objects is useful functionality in the user interface, but it provides little functionality by itself in the Visual Basic environment. For a powerful way to deal with groups of diagram objects in Visual Basic, see the ObjectRange object.

The combination of the Group and ObjectRange objects provides the functionality of the Arrange—Group menu item from the iGrafx Professional interface.

For more information on grouping shapes, and what you can do with the group, refer to the iGrafx Professional User’s Guide.

Properties, Methods, and Events

All of the properties, methods, and events for the Group object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Ungroup	
DiagramObject		
ObjectRange		
Parent		

DiagramObject Property

Syntax *Group*.**DiagramObject**

Data Type DiagramObject object (read-only, See Object Properties)

Description The DiagramObject property returns the DiagramObject object associated with the specified Group object. This property provides access to the properties, methods, and events at the DiagramObject level (a Group object is one of several types of DiagramObject objects).

Example The following example creates an ObjectRange containing three objects. It then uses the ObjectRange property to create a Group object. Finally it uses the DiagramObject property to set and display the Group object's object name.

```
' Dimension the variables
Dim igxDiagram As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
Dim igxGroup As Group
' Create diagram and shape objects
Set igxDiagram = Application.ActiveDiagram
Set igxShape1 = igxDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = igxDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = igxDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Create an ObjectRange object in the diagram
Set igxObjectRange = igxDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Group all the objects in the ObjectRange
MsgBox "Click OK to group the objects"
Set igxGroup = igxObjectRange.Group
igxGroup.DiagramObject.ObjectName = "Group 1"
' Display the Group Object name using the DiagramObject property
MsgBox "The Group object name is: " & igxGroup.DiagramObject.ObjectName
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button Group object,JI(`igrafxrf.HLP',`Group_Object')}

ObjectRange Property

Syntax *Group*.**ObjectRange**

Data Type ObjectRange object (read-only, See Object Properties)

Description The ObjectRange property returns an ObjectRange object that contains all the objects in the Group object.

Example The following example creates three shapes in the active diagram, and creates an ObjectRange that contains the first and third shapes. Then a Group object is made from the ObjectRange by using its Group property. A second ObjectRange object is then populated by using the Group object's ObjectRange property, then this second ObjectRange is moved down, rotated right, and filled with a solid red. Next, the Group object is ungrouped, and all objects removed from the two ObjectRange variables. A new group is created from the ObjectRange property of the DiagramObjects collection (which contains all objects in the diagram). The fill of this new Group is then changed to blue.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange1 As ObjectRange
Dim igxObjectRange2 As ObjectRange
Dim igxGroup As Group
' Create several shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Create an ObjectRange object in the diagram
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange object
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape3.DiagramObject
' Group all the objects in the ObjectRange
MsgBox "Click OK to group the objects"
Set igxGroup = igxObjectRange1.Group
MsgBox "The ObjectRange property of the Group object " _
    & Chr(13) & "contains " & igxGroup.ObjectRange.Count _
    & " objects."
igxGroup.DiagramObject.ObjectName = "Group 1"
MsgBox "Now click OK to use the Group to create a new ObjectRange."
Set igxObjectRange2 = igxGroup.ObjectRange
' Move the new object range down and rotate the objects its contains
' 90 degrees to the right, and add a red fill
MsgBox "New ObjectRange created. Click OK to move, rotate and fill " _
    & Chr(13) & "this new ObjectRange."
igxObjectRange2.CenterY = 1440 * 5
igxObjectRange2.Rotate (ixRotateRight)
igxObjectRange2.FillFormat.FillColor = vbRed
MsgBox "Click OK to ungroup the objects and remove all objects " _
   & Chr(13) & "from both object ranges"
' Ungroup the Group object, and remove all objects from
' the two ObjectRange variables
igxGroup.Ungroup
igxObjectRange1.RemoveAll
```

```
igxObjectRange2.RemoveAll
' Make a Group from all the objects in the diagram
MsgBox "Now make another Group from all the objects, " _
    & " and change the fill to blue. Click OK to continue."
MsgBox "The ObjectRange property of the DiagramObjects object " _
    & Chr(13) & "contains " & _
    ActiveDiagram.DiagramObjects.ObjectRange.Count _
    & " objects."
Set igxGroup = ActiveDiagram.DiagramObjects.ObjectRange.Group
' Change the fill color of the group
igxGroup.ObjectRange.FillFormat.FillColor = vbBlue
MsgBox "View the result."
```

See Also

ObjectRange object

iGrafx API Object Hierarchy

{button Group object,JI(`igrafxrf.HLP',`Group_Object')}

Ungroup Method

Syntax *Group*.**Ungroup**

Description The Ungroup method ungroups the objects in a Group object, emptying the Group object. This is the same as selecting the Arrange->Ungroup menu item from the iGrafx Professional interface.

Example The following example creates three shapes in the active diagram. It then creates an ObjectRange, and adds all three shapes to it. The ObjectRange is then made into a group. The Ungroup method is then applied. Then, to show the relationship between a Group and an ObjectRange, a new ObjectRange is set from the ObjectRange property of the Group object. Then the fill color of the new ObjectRange is set to red, except that this does not work because ObjectRange1 is actually empty because of the Ungroup method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange1 As ObjectRange
Dim igxObjectRange2 As ObjectRange
Dim igxGroup As Group
' Create several shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Create an ObjectRange object in the diagram
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange object
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.Add igxShape3.DiagramObject
' Group all the objects in the ObjectRange
MsgBox "Click OK to group the objects"
Set igxGroup = igxObjectRange1.Group
MsgBox "Group created.  Now click OK to Ungroup the group."
igxGroup.Ungroup
MsgBox "Now click OK to use the Group to create a new ObjectRange."
Set igxObjectRange2 = igxGroup.ObjectRange
MsgBox "Click OK to apply a fill color to the new ObjectRange."
igxObjectRange2.FillFormat.FillColor = vbRed
MsgBox "Nothing happened because the ObjectRange is empty. It was" _
    & Chr(13) & "derived from a Group object that had been Ungrouped."
```

{button Group object,JI(`igrafxrf.HLP',`Group_Object')}

Layer Object

The Layer object controls the use of “layers”. Essentially, this object provides the same functionality as using the Layer Manager from the iGrafx Professional interface. The Layer object is subordinate to the ??? object, and is accessed through the Layers collection object. Most operations you perform with the Layer object are straightforward, such as making a layer visible or printable. The Layer object also provides access to all of the DiagramObject objects on the layer through the DiagramObjects property.

If you require more information about layers and their use within iGrafx Professional, refer to the iGrafx Professional User’s Guide.

The following code demonstrates how to access the active layer in the active diagram.

```
' Dimension the variables
Dim igxLayer As Layer
' Set our layer variable to the current active layer
Set igxLayer = ActiveDiagram.ActiveLayer
MsgBox "Our Visual Basic layer variable has been set" _
    & Chr(13) & "to the current active layer."
```

Important Notes about Using Layers

- Shapes drawn on different layers can be connected with connector lines, provided that all layers are unlocked.
- Locking a layer means that no modifications can be made to that layer through the user interface. Modification can still be made using VBA.
- A layer cannot be locked if it is the currently active layer.
- Layer order is maintained in the Layers collection object, and is viewed through the interface by tabs at the bottom of a diagram window.

Properties, Methods, and Events

All of the properties, methods, and events for the Layer object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Activate	
Index	Delete	
Locked	MoveDown	
Name	MoveUp	
ObjectRange		
Parent		
Printable		
Visible		

Related Topics

Layers object

iGrafx API Object Hierarchy

Activate Method

Syntax *Layer.***Activate**

Description The Activate method makes the specified layer the current layer. For example, assume you have two layers named Layer1 and Layer2, and Layer1 is the currently active layer. Activating Layer2 is done with the following statement:

```
Layer2.Activate
```

Example The following example demonstrates how to activate layers on the current diagram.

```
' Dimension the variables
Dim igxLayer1 As Layer
Dim igxLayer2 As Layer
Dim igxShape As Shape
' Set a variable to Layer 1, the current active layer
Set igxLayer1 = ActiveDiagram.ActiveLayer
' Create a new layer in the document
Set igxLayer2 = ActiveDiagram.Layers. _
   Add("Layer 2", False, True, True)
igxLayer1.Activate
MsgBox "Click OK to Activate Layer 2"
' Activate Layer 2
igxLayer2.Activate
MsgBox "New click OK to Activate Layer 1"
' Activate Layer 1
igxLayer1.Activate
MsgBox "Click OK to continue"
```

{button Layer object,JI(`igrafxrf.HLP',`Layer_Object')}

Index Property

Syntax *Layer*.**Index**

Data Type Integer (read-only)

Description The Index property returns the Index value of the specified Layer object. This value is the position of the layer within the Layers collection, which also represents the ordering of layers in a diagram.

Example The following example sets up a document with two layers, and then displays the Index property of each layer.

```
' Dimension the variables
Dim igxLayer1 As Layer
Dim igxLayer2 As Layer
' Set the variable for Layer 1
Set igxLayer1 = ActiveDiagram.Layers.Item(1)
' Create Layer 2, and set the variable
Set igxLayer2 = ActiveDiagram.Layers. _
   Add("Layer 2", False, True, True)
' Display the Layer Index numbers
MsgBox "The Index of Layer 1 is: " & igxLayer1.Index
MsgBox "The Index of Layer 2 is: " & igxLayer2.Index
MsgBox "Click OK to continue"
```

{button Layer object,JI(`igrafxrf.HLP',`Layer_Object')}

Locked Property

Syntax *Layer*.**Locked**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Locked property specifies whether the layer is locked. Locking a layer only locks items from the user interface. Visual Basic can access and modify items on locked layers.

Locking a layer means that no modifications can be made to that layer from the user interface. A layer cannot be locked if it is the currently active layer. Also, a layer cannot be deleted if it is the only unlocked visible layer.

Example The following example modifies a diagram so it has two layers. A shape is placed on Layer 1, and then Layer 1 is locked. Then the fill color of the shape is changed to blue to show that locking a layer does not prevent changes to the Shape object by using VBA. However, return to the user interface and attempt to change the fill of the shape, and verify that it cannot be changed.

```
' Dimension the variables
Dim igxLayer1 As Layer
Dim igxLayer2 As Layer
Dim igxShape As Shape
' Set a variable to Layer 1, the current active layer
Set igxLayer1 = ActiveDiagram.ActiveLayer
' Create a new layer in the document
Set igxLayer2 = ActiveDiagram.Layers.Add _
    ("Layer 2", False, True, True)
' Activate Layer 1
MsgBox "Click OK to Activate Layer 1, create a shape, and lock the layer."
igxLayer1.Activate
' Create a new shape in Layer 1
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Activate Layer 2
igxLayer2.Activate
' Lock Layer 1
igxLayer1.Locked = True
MsgBox "Layer 1 has been locked"
' Try to change a shape on the locked layer
igxShape.FillColor = vbBlue
MsgBox "The shape has been changed to blue using Visual Basic" _
    & Chr(13) & "even though the shape is on a Locked layer."
```

{button Layer object,JI(`igrafxrf.HLP',`Layer_Object')}

MoveDown Method

Syntax *Layer*.**MoveDown**

Description The MoveDown method moves a layer down in relation to the other layers, decreasing the layer's index by one. The ordering of layers can be important for certain tasks or display of information. The ordering of layers is maintained in the Layers collection object. The MoveDown method updates the Layer.Index property to reflect the new order, and rearranges the layer tabs on the document interface.

Example The following example demonstrates how to move down the second layer of the current diagram.

```
' Dimension the variables
Dim igxLayer1 As Layer
Dim igxLayer2 As Layer
Dim igxShape As Shape
' Set a variable to Layer 1, the current active layer
Set igxLayer1 = ActiveDiagram.ActiveLayer
' Create a new layer in the document
Set igxLayer2 = Application.ActiveDiagram.Layers. _
   Add("Layer 2", False, True, True)
' Display the current index values for the two layers
MsgBox "The Index of Layer 1: " & igxLayer1.Index & _
   Chr(13) & "The Index of Layer 2: " & igxLayer2.Index
' Perform the MoveDown method on Layer 2
MsgBox "Click OK to perform the Layer 2 MoveDown method."
igxLayer2.MoveDown
MsgBox "The Index of Layer 1: " & igxLayer1.Index & _
   Chr(13) & "The Index of Layer 2: " & igxLayer2.Index _
   & Chr(13) & "The index's were changed to reflect the new order." _
   & Chr(13) & "Also, the layer tabs in the diagram were rearranged."
```

{button Layer object,JI(`igrafxrf.HLP',`Layer_Object')}

MoveUp Method

Syntax *Layer*.**MoveUp**

Description The MoveUp method moves a layer up in relation to the other layers, increasing the layer's index by one. The ordering of layers can be important for certain tasks or display of information. The ordering of layers is maintained in the Layers collection object. The MoveUp method updates the Layer.Index property to reflect the new order, and rearranges the layer tabs on the document interface.

Example The following example demonstrates how to move up the second layer of the current diagram.

```
' Dimension the variables
Dim igxLayer1 As Layer
Dim igxLayer2 As Layer
Dim igxShape As Shape
' Set a variable to Layer 1, the current active layer
Set igxLayer1 = ActiveDiagram.ActiveLayer
' Create a new layer in the document
Set igxLayer2 = ActiveDiagram.Layers. _
   Add("Layer 2", False, True, True)
' Display the current index values for the two layers
MsgBox "The Index of Layer 1: " & igxLayer1.Index & _
   Chr(13) & "The Index of Layer 2: " & igxLayer2.Index
' Perform the MoveUp method on Layer 1
MsgBox "Click OK to perform the Layer 1 MoveUp method."
IgxLayer1.MoveUp
MsgBox "The Index of Layer 1: " & igxLayer1.Index & _
   Chr(13) & "The Index of Layer 2: " & igxLayer2.Index _
   & Chr(13) & "The index's were changed to reflect the new order." _
   & Chr(13) & "Also, the layer tabs in the diagram were rearranged."
```

{button Layer object,JI(`igrafxrf.HLP',`Layer_Object')}

ObjectRange Property

Syntax *Layer*.**ObjectRange**

Data Type ObjectRange object (read-only, See Object Properties)

Description The ObjectRange property returns the ObjectRange object for the specified Layer object. This object contains all of the various “diagram objects” that are contained on the layer.

Example The following example demonstrates how to assign a layer’s ObjectRange to an ObjectRange variable, and modify the shapes in the new ObjectRange.

```
' Dimension the variables
Dim igxLayer As Layer
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Set the layer variable to the current layer
Set igxLayer = ActiveDiagram.ActiveLayer
' Create 3 shapes and assign the shape variables to the 3 Shape objects.
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Set the igxObjectRange variable to the layer's ObjectRange
MsgBox "Click OK to obtain the layers ObjectRange"
Set igxObjectRange = igxLayer.ObjectRange
MsgBox "The ObjectRange property of the Layer object " _
    & Chr(13) & "contains " & igxObjectRange.Count _
    & " objects."
' Set the FillColor for the shapes in the ObjectRange to blue
MsgBox "Click OK to apply a fill color to this layer's ObjectRange"
igxObjectRange.FillFormat.FillColor = vbBlue
MsgBox "All the items on the layer have been changed."
```

See Also ObjectRange object

iGrafx API Object Hierarchy

{button Layer object,JI(`igrafxrf.HLP',`Layer_Object')}

Printable Property

Syntax *Layer*.**Printable**[= {**True | False**}]

Data Type Boolean (read/write)

Description The Printable property specifies whether a layer can be printed. Setting this properrty to False means that any objects on that layer are not printed. This property provides a way for diagrams to include information that is useful while creating or editing a diagram, but is not useful when the diagram is printed. This property provides the same functionality as checking the “Printable” checkbox in the Layer Manager through the user interface.

Example The following example demonstrates how to make the layers of the current diagram printable, and not printable.

```
' Dimension the variables
Dim igxLayer1 As Layer
Dim igxLayer2 As Layer
' Set igxLayer1 variable to diagram layer 1
Set igxLayer1 = ActiveDiagram.Layers.Item(1)
' Set igxLayer2 to a new layer
Set igxLayer2 = ActiveDiagram.Layers. _
   Add("Layer 2", False, True, True)
' Change the Printable property of each layer
MsgBox "Click OK to modify the Printable property of each layer."
igxLayer1.Printable = False
igxLayer2.Printable = True
MsgBox "Go to the diagram, right click on each layer tab" _
    & Chr(13) & "and view the Printable attribute of each layer." _
    & Chr(13) & "Layer 2 should have Printable checked, and" _
    & Chr(13) & "Layer 1 should have Printable not checked."
```

{button Layer object,JI(`igrafxrf.HLP',`Layer_Object')}

Layers Object

The Layers object is a collection of individual Layer objects. A Layers collection is only associated with and accessible from the Diagram object. Its purpose is to store and provide access to the individual Layer objects that have been created for a diagram.

The Layers object provides the following functionality:

- The ability to access any Layer objects that have been created for a particular Diagram object.
- The ability to determine how many Layer objects are in the collection.
- The ability to edit all of the layers that currently exist for a specific diagram.
- The ability to add a new layer to a diagram.

The Layers object also maintains the ordering of the individual Layer objects (important for the Layer.MoveUp and Layer.MoveDown methods).

Properties, Methods, and Events

All of the properties, methods, and events for the Layers object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
EditAllLayers		
Parent		

Related Topics

Layer Object

iGrafx API Object Hierarchy

Add Method

Syntax *Layers*.**Add** (*Name* As String, [*Locked* As Boolean = False], [*Visible* As Boolean = True], [*Printable* As Boolean = True]) As Layer

Description The Add method adds a Layer object to the Layers collection. The Name argument specifies the name of the layer. The added layer is appended to the end of the Layers collection in the layer order. Therefore, if there is a specific place in the layer order, you need to know how many layers are currently in the collection, and use the Layer.MoveUp or Layer.MoveDown methods to appropriately position the added layer.

The *Name* argument specifies the name of the Layer object. The name appears on the Layers tab in the user interface.

The *Locked* argument specifies whether the Layer object is locked. If set to True, the Layer is locked. If set to False, the Layer is not locked (default.)

The *Visible* argument specifies whether the Layer is visible. If set to True, the Layer and it's objects are visible (default). If set to False, the Layer and the objects on it are not visible.

The *Printable* argument specifies whether the Layer is printable. If set the True, the objects on the Layer can be printed. If set to False, the objects on the Layer are not printed when the diagram is printed.

Example The following example demonstrates how to add a layer to the current diagram.

```
' Dimension the variables
Dim igxLayer As Layer
' Create a new layer in the document
Set igxLayer = Application.ActiveDiagram.Layers. _
   Add("Layer 2", False, True, True)
Application.RefreshUI
MsgBox "Layer 2 added to the Diagram."
```

{button Layers object,JI(`igrafxrf.HLP',`Layers_Object')}

EditAllLayers Property

Syntax *Layers*.**EditAllLayers**[= {**True | False**}]

Data Type Boolean (read/write)

Description The EditAllLayers property specifies whether all currently defined layers are accessible at once. If set to True, editing of objects according to layers is not in effect, and all DiagramObjects can be accessed by the user without changing layers. If set to False, editing by layers is in effect, and the user must change layers to access a DiagramObject on its particular layer.

Example The following example sets the EditAllLayers property to True, which allows the user to access all objects in the diagram at once, without having to change layers.

```
' Dimension the variables
Dim igxLayer As Layer
' Create a new layer in the document
Set igxLayer = Application.ActiveDiagram.Layers. _
    Add("Layer 2", False, True, True)
' Switch on the EditAllLayers property
Application.ActiveDiagram.Layers.EditAllLayers = True
MsgBox "Layer 2 added to the Diagram, and All Layers" _
   & Chr(13) & "are live for editing."
```

{button Layers object,JI(`igrafxrf.HLP',`Layers_Object')}

Item Method

Syntax *Layers*.**Item**(*Index* As Integer) As Layer

Description The Item method returns the Layer object at the specified *Index* from the Layers collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Layer. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is important to use error trapping before attempting to use the Item method.

Example The following example demonstrates how to use the Item Method to access the second layer in the current document, and extract the name of the layer.

```
' Dimension the variables
Dim igxLayer As Layer
Dim strLayerName As String
' Create a new layer in the document
Set igxLayer = Application.ActiveDiagram.Layers. _
   Add("Layer 2", False, True, True)
' Get the name of the new layer
strLayerName = Application.ActiveDiagram.Layers.Item(2).Name
' Display the name of the new layer
MsgBox "A new layer has been added to the Diagram." _
   & Chr(13) & "The name of the new layer is: " & strLayerName
```

{button Layers object,JI(`igrafxrf.HLP',`Layers_Object')}

Link Object

The Link object allows a shape to be linked to a diagram or a file. A link adds a menu item to a shape's context menu that can be clicked by the user, or executed from Visual Basic. The Link object is subordinate to the Shape object, through the Links collection object. Only shapes can have links. Furthermore, any individual shape can contain multiple links, but only one sub-process link.

Diagram links are used to link one diagram to another, and optionally accumulate custom data from the linked diagram. Diagram links rediect the flow of transactions and entities. When a process or Entity is run, they follow any links they encounter.

There is one special type of diagram link called a *Sub-process link*. A Sub-process link specifies that the linked diagram is a *Sub-process*. When a transaction or entity encounters a Sub-process link, it follows the link and completes the sub-process, but then returns back to the original diagram and shape, and continues it's progress. A shape can have only one Sub-process link.

File links connect shapes to files. A file link adds a menu item to the shape's context menu. The menu item can be clicked to launch the file. Also, the file can be launched from Visual Basic using the Link.Execute method.

Properties, Methods, and Events

All of the properties, methods, and events for the Link object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
AccumulateData	Delete	
Application	Execute	
Description		
IsSubProcess		
Key		
Parent		
StartPointName		
Target		
Type		

Related Topics

Links object

iGrafx API Object Hierarchy

AccumulateData Property

Syntax *Link*.**AccumulateData**[= {**True | False**}]

Data Type Boolean (read/write)

Description The AccumulateData property specifies whether the shape that contains the specified Link object accumulates custom data from the link's target diagram. If set to True, the shape containing the link accumulates custom data from the target. If set to False, the shape containing the link does not accumulate custom data from the target. The custom data is stored in the Shape's CustomDataValues collection according to the AccumulationMethod specified by the parent CustomDataDefinition of each CustomDataValue.

Example The following example sets up two diagrams with shapes, and adds a CustomDataDefinition to the document. Diagram A has a link to Diagram B. A CustomDataValue is changed on Diagram B. Then the UpdateAll method is used to reflect the change in the Legend in Diagram A.

```
' Dimension the variables
Dim igxDiagramA As Diagram
Dim igxDiagramB As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxDataDef As CustomDataDefinition
Dim igxLink As Link
' Create two diagrams
Set igxDiagramA = ActiveDocument.Diagrams.Add("Diagram A")
Set igxDiagramB = ActiveDocument.Diagrams.Add("Diagram B")
' Add shapes to Diagram A
Set igxShape1 = igxDiagramA.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = igxDiagramA.DiagramObjects.AddShape(1440 * 3, 1440)
igxDiagramA.DiagramObjects.AddLegend 3000, 3000
' Add shapes to Diagram B
Set igxShape3 = igxDiagramB.DiagramObjects.AddShape(1440, 1440)
Set igxShape4 = igxDiagramB.DiagramObjects.AddShape(1440 * 3, 1440)
igxShape3.StartPointName = "StartB"
' Add a CustomDataDefinition to the document
ActiveDocument.CustomDataDefinitions.Add _
   "Cost", ixCustomDataFormatCurrencyBase
' Set the CustomDataValues
igxShape1.DiagramObject.CustomDataValues.Item(1).Value = 10
igxShape2.DiagramObject.CustomDataValues.Item(1).Value = 10
igxShape3.DiagramObject.CustomDataValues.Item(1).Value = 10
igxShape4.DiagramObject.CustomDataValues.Item(1).Value = 10
' Add a link from Diagram A to Diagram B
Set igxLink = igxShape2.Links.AddDiagramLink("Diagram B")
' Have the link accumulate data
igxLink.AccumulateData = True
' Activate Diagram A so we can see it
igxDiagramA.ActivateDiagram
' Change a CustomDataValue on Diagram B
MsgBox "Click to change a CustomDataValue on Diagram B"
igxShape3.DiagramObject.CustomDataValues.Item(1).Value = 20
' Update the accumulation data
```

```
igxShape2.DiagramObject.CustomDataValues.UpdateAll
MsgBox "Click OK to continue"
```

{button Link object,JI(`igrafxrf.HLP',`Link_Object')}

Description Property

Syntax *Link*.**Description**

Data Type String (read/write)

Description The Description property specifies a text string that describes the link. The Description string is how a link appears in a context menu when you right-click on a shape. It is the same string that is edited in a shape's Properties->General Tab->Links->Edit… option.

Example The following example creates a shape, a diagram, and a link to a file. Then the new Link object's Description property is set to a string, and displayed in a Message Box.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxLink As Link
' Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxLink variable to a new Link object
Set igxLink = igxShape.Links.AddDiagramLink("NewDiagram")
' Set the Description Property to a string
igxLink.Description = "This is a link to NewDiagram."
MsgBox "The Link's Description Property is set to:" & igxLink.Description
```

{button Link object,JI(`igrafxrf.HLP',`Link_Object')}

Execute Method

Syntax *Link*.**Execute**

Description The Execute Method causes a link to be executed. If it's a Diagram link, the process proceeds to the named diagram. If it's a File link, the file executes.

Example The following example creates a shape, a diagram, and a link to a file. Then the new Link is executed. In this example, it's a File link to the WordPad program, which starts a WordPad session.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxLink As Link
' Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxLink variable to a new Link object
Set igxLink = igxShape.Links.AddFileLink("wordpad.exe")
' Execute the link
MsgBox "Click OK to execute the link, which starts WordPad."
igxLink.Execute
```

{button Link object,JI(`igrafxrf.HLP',`Link_Object')}

IsSubProcess Property

Syntax *Link*.**IsSubProcess**[= {**True | False**}]

Data Type Boolean (read/write)

Description The IsSubProcess property specifies whether the specified Link object is a sub-process link. If it is a sub-process link, all Entities that encounter the link jump to the sub-process diagram, return, and continue their progress. Sub-processes must be contained in a separate diagram; that is, they cannot be on the same diagram as another process.

Example The following example creates two diagrams. It links Diagram A to Diagram B, and makes Diagram B a sub-process. It then runs an Entity and follows the Entity's progress through the sub-process diagram.

```
Private Sub Main()
    ' Dimension the variables
    Dim igxDiagramA As Diagram
    Dim igxDiagramB As Diagram
    Dim igxShape1 As Shape
    Dim igxShape2 As Shape
    Dim igxShape3 As Shape
    Dim igxShape4 As Shape
    Dim igxConnector1 As ConnectorLine
    Dim igxConnector2 As ConnectorLine
    Dim igxLink As Link
    ' Add two diagrams
    Set igxDiagramA = ActiveDocument.Diagrams.Add("Diagram A")
    igxDiagramA.DiagramObjects.AddTextObject 3000, 500, , , "DIAGRAM A"
    Set igxDiagramB = ActiveDocument.Diagrams.Add("Diagram B")
    igxDiagramB.DiagramObjects.AddTextObject 3000, 500, , , "DIAGRAM B"
    ' Add shapes to Diagram A
    Set igxShape1 = igxDiagramA.DiagramObjects.AddShape(1440, 1440)
    Set igxShape2 = igxDiagramA.DiagramObjects.AddShape(1440 * 3, 1440)
    Set igxConnector1 = igxDiagramA.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
    ' Add shapes to Diagram B
    Set igxShape3 = igxDiagramB.DiagramObjects.AddShape(1440, 1440)
    Set igxShape4 = igxDiagramB.DiagramObjects.AddShape(1440 * 3, 1440)
    Set igxConnector2 = igxDiagramB.DiagramObjects.AddConnectorLine _
       (ixRouteDirect, , igxShape3, ixDirEast, , , , igxShape4, ixDirWest)
    ' Add a link from Diagram A to Diagram B
    Set igxLink = igxShape2.Links.AddDiagramLink("Diagram B")
    ' Make the link a sub process
    igxLink.IsSubProcess = True
    ' Add an Entity to the document
    ThisDocument.Entities.Add "MyEntity", igxShape1
    ' Run the Entity
    ThisDocument.Entities.Item(1).Run
End Sub

Private Sub AnyShape_EntityExecute(ByVal Entity As IGrafx2.IXEntity)
    ' Pause each time the Entity enters a shape
    MsgBox "Continue Entity"
End Sub
```

{button Link object,JI(`igrafxrf.HLP',`Link_Object')}

Key Property

Syntax *Link*.**Key**

Data Type IxKeyModifier enumerated constant (read/write)

Description The Key property specifies a keyboard key combination that, combined with a mouse double-click on a shape, activates a specific link.

The IxKeyModifier constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixModifierNone
1	ixModifierCtrl
2	ixModifierShift
3	ixModifierAlt
4	ixModifierCtrlShift
5	ixModifierCtrlAlt
6	ixModifierAltShift
7	ixModifierCtrlAltShift

Example The following example creates a shape, a diagram, and a link to a file. Then the new Link object's Key property is set to an enumerated constant to set the hot key sequence for this link.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxLink As Link
'Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxLink variable to a new Link object
Set igxLink = igxShape.Links.AddFileLink("wordpad.exe")
' Set a hotkey sequence for this link
igxLink.Key = ixModifierCtrl
MsgBox "A Link Key has been added to the shape." _
    & Chr(13) & "To test it, click OK, and then go back to the diagram." _
    & Chr(13) & "In the diagram hold down the Ctrl key, and " _
    & Chr(13) & "double-click the shape to execute the Link."
```

{button Link object,JI(`igrafxrf.HLP',`Link_Object')}

StartPointName Property

Syntax *Link*.**StartPointName**

Data Type String (read/write)

Description The StartPointName property specifies the name of a start point for the specified Link object. If the target diagram has more than one start point, this property specifies which start point the Link should use.

If the target diagram only has one start point, the Link uses that start point automatically; therefore, specifying the start point name with this property is not necessary, but is advisable in case other start points are subsequently added to the diagram. If you do not explicitly specify a start point name for the link, then the link uses the first start point it finds in the target diagram, and this may not be the desired effect. Also, not specifying a start point name explitcitly means that this property is empty, and then there is no way to check or verify which start point was used.

Example The following example creates two diagrams, and links one diagram to the other. Diagram2 has two start points, so the Link's start point is set to "StartPoint2" using the StartPointName property.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxLink As Link
' -------- This section creates two complete diagrams ----------
' Create Diagram 1
Set igxDiagram1 = ThisDocument.Diagrams.Add("Example Process A")
Set igxShape1 = igxDiagram1.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = igxDiagram1.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector1 = igxDiagram1.DiagramObjects.AddConnectorLine _
  (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
igxShape1.StartPointName = "StartPoint"
igxDiagram1.LinkIndicatorStyle.Style = ixLinkIcon
' Create Diagram 2
Set igxDiagram2 = ThisDocument.Diagrams.Add("Example Process B")
Set igxShape3 = igxDiagram2.DiagramObjects.AddShape(1440, 1440)
Set igxShape4 = igxDiagram2.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector2 = igxDiagram2.DiagramObjects.AddConnectorLine _
  (ixRouteDirect, , igxShape3, ixDirEast, , , , igxShape4, ixDirWest)
igxShape3.StartPointName = "StartPoint1"
igxShape4.StartPointName = "StartPoint2"
igxDiagram2.LinkIndicatorStyle.Style = ixLinkIcon
igxDiagram1.ActivateDiagram
' ---------------------------------------------------------
' Add a link to Shape2 that links Diagram1 to Diagram2
Set igxLink = igxShape2.Links.AddDiagramLink("Example Process B")
' Set the Link's start point on the target Diagram
igxLink.StartPointName = "StartPoint2"
MsgBox "The Link's start point is: " & igxLink.StartPointName
```

{button Link object,JI(`igrafxrf.HLP',`Link_Object')}

Target Property

Syntax *Link*.**Target**

Data Type String (read/write)

Description The Target property specifies the destination of the link. If the link is to a diagram, then you can get or set the diagram name. If the link is to a file, then you can get or set the file name. Providing absolute file names is recommended. Note that a link to a Web page (a URL) is considered a file name.

For Diagram links, the Target property specifies the name of the Diagram, but it does not provide information on which start point the Link links to, which is specified by the StartPointName property.

The StartPointName property is used for links that refer to other shapes.

Example The following example creates two diagrams, and links them together. The Link object's Target property is then displayed.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxLink As Link
' -------- This section creates two complete diagrams ----------
' Create Diagram 1
Set igxDiagram1 = ThisDocument.Diagrams.Add("Example Process A")
Set igxShape1 = igxDiagram1.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = igxDiagram1.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector1 = igxDiagram1.DiagramObjects.AddConnectorLine _
  (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
igxShape1.StartPointName = "StartPoint"
igxDiagram1.LinkIndicatorStyle.Style = ixLinkIcon
' Create Diagram 2
Set igxDiagram2 = ThisDocument.Diagrams.Add("Example Process B")
Set igxShape3 = igxDiagram2.DiagramObjects.AddShape(1440, 1440)
Set igxShape4 = igxDiagram2.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector2 = igxDiagram2.DiagramObjects.AddConnectorLine _
  (ixRouteDirect, , igxShape3, ixDirEast, , , , igxShape4, ixDirWest)
igxShape3.StartPointName = "StartPoint1"
igxShape4.StartPointName = "StartPoint2"
igxDiagram2.LinkIndicatorStyle.Style = ixLinkIcon
igxDiagram1.ActivateDiagram
' ------------------------------------------------------------
' Add a link to Shape2 that links Diagram1 to Diagram2
Set igxLink = igxShape2.Links.AddDiagramLink("Example Process B")
' Set the Link's start point on the target Diagram
igxLink.StartPointName = "StartPoint2"
MsgBox "The Link's target is: " & igxLink.Target
```

See Also

Shape object

iGrafx API Object Hierarchy

{button Link object,JI(`igrafxrf.HLP',`Link_Object')}

Type Property

Syntax *Link*.**Type**

Data Type IxLinkType enumerated constant (read-only)

Description The Type property returns the type of the specified Link object.

The IxLinkType constant defines the valid values for this property, which are listed in the following table.

Value	Name of Constant
0	ixLinkToFile
1	ixLinkToDiagram
2	ixLinkToShape

Note The ixLinkToShape type is no longer used in iGrafx Professional diagrams. It is included for backward compatibility with Micrografx Flowcharter 7 files.

Example The following example creates two diagrams and links them. Then the new Link object's Type property is checked and the results displayed.

```
' Dimension the variables
Dim igxDiagram1 As Diagram
Dim igxDiagram2 As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxLink As Link
' -------- This section creates two complete diagrams ----------
' Create Diagram 1
Set igxDiagram1 = ThisDocument.Diagrams.Add("Example Process A")
Set igxShape1 = igxDiagram1.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = igxDiagram1.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector1 = igxDiagram1.DiagramObjects.AddConnectorLine _
  (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
igxShape1.StartPointName = "StartPoint"
igxDiagram1.LinkIndicatorStyle.Style = ixLinkIcon
' Create Diagram 2
Set igxDiagram2 = ThisDocument.Diagrams.Add("Example Process B")
Set igxShape3 = igxDiagram2.DiagramObjects.AddShape(1440, 1440)
Set igxShape4 = igxDiagram2.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector2 = igxDiagram2.DiagramObjects.AddConnectorLine _
  (ixRouteDirect, , igxShape3, ixDirEast, , , , igxShape4, ixDirWest)
igxShape3.StartPointName = "StartPoint1"
igxShape4.StartPointName = "StartPoint2"
igxDiagram2.LinkIndicatorStyle.Style = ixLinkIcon
igxDiagram1.ActivateDiagram
' ---------------------------------------------------------
' Add a link to Shape2 that links Diagram1 to Diagram2
Set igxLink = igxShape2.Links.AddDiagramLink("Example Process B")
```

```
Select Case igxLink.Type
    Case 0
        MsgBox "The Link Type is a File Link"
    Case 1
        MsgBox "The Link Type is a Diagram Link"
    Case 2
        MsgBox "The Link Type is a Shape Link"
End Select
```

{button Link object,JI(`igrafxrf.HLP',`Link_Object')}

Links Object

The Links object is a collection of Link objects. A Links collection is associated with each Shape object. Its purpose is to store and provide access to the individual Link objects that have been created for a shape.

The Links object provides the following functionality:

- The ability to access any Link objects that have been created for a particular Shape object, including sub-process links.
- The ability to determine how many Link objects are currently in the collection.
- The ability to delete all of the links that exist for a specific shape. You can delete a single Link object through the Link object.
- The ability to add a new link of a specific type (a link to a file, a diagram, or another shape) to a Shape object.

Properties, Methods, and Events

All of the properties, methods, and events for the Links object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	AddDiagramLink	
Count	AddFileLink	
Parent	DeleteAll	
SubProcessLink	Item	

Related Topics

Link Object

iGrafx API Object Hierarchy

AddDiagramLink Method

Syntax *Links*.**AddDiagramLink**(*DiagramName* As String) As Link

Description The AddDiagramLink method adds a new “diagram” link to the Links collection. This method specifically adds to a shape, a link from the shape to a diagram. If the *DiagramName* argument is not a valid diagram, an error is returned.

Example The following example creates a shape, a diagram, and a link to a diagram. Then the new Link object’s Description property is set to a string.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxLink As Link
' Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
' Set igxLink variable to a new Link object
Set igxLink = igxShape.Links.AddDiagramLink("NewDiagram")
' Set the Description Property to a string
igxLink.Description = "This is a link to another diagram."
MsgBox "A new Diagram Link was added to the shape." _
    & Chr(13) & "You can check the link by opening the shape's" _
    & Chr(13) & "Property Dialog, and going to the General Tab."
```

{button Links object,JI(`igrafxrf.HLP',`Links_Object')}

AddFileLink Method

Syntax *Links*.**AddFileLink**(*Path* As String) As Link

Description The AddFileLink method adds a new “file” link to the Links collection. This method specifically adds to a shape, a link from the shape to a file. If the *Path* argument is not a valid file, an error is returned.

Example The following example creates a shape, a diagram, and a link to a file. Then the new Link object’s Description property is set to a string.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxLink As Link
' Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxLink variable to a new Link object
Set igxLink = igxShape.Links.AddFileLink("wordpad.exe")
' Set the Description Property to a string
igxLink.Description = "This link will start WordPad."
MsgBox "A file link to WordPad was created.  After clicking OK," _
    & Chr(13) & "go back to the diagram, and right-click the shape to" _
    & Chr(13) & "find the new File Link that starts WordPad."
```

{button Links object,JI(`igrafxrf.HLP',`Links_Object')}

DeleteAll Method

Syntax *Links*.**DeleteAll**

Description The DeleteAll method deletes all links from the Links collection for the specified Shape object. CAUTION: Be sure this is what you want to do before using this method, because once the links are deleted, they cannot be recovered.

Example The following example creates a shape, two diagrams, and two links to the diagrams. Then the new Link object's Description properties are set to strings.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxLink1 As Link
Dim igxLink2 As Link
'Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxLink1 variable to a new Link object
Set igxLink1 = igxShape.Links.AddDiagramLink("NewDiagram")
' Set the Description Property to a string
igxLink1.Description = "This is a link to a diagram."
' Set igxLink2 variable to a new Link object
Set igxLink2 = igxShape.Links.AddDiagramLink("NewDiagram2")
' Set the Description Property to a string
igxLink2.Description = "This is a link to another diagram."
MsgBox ("Two new links have been created in the shape." _
   & Chr(13) & "Click OK to delete both links using the" _
   & " DeleteAll Method.")
' Delete all links
igxShape.Links.DeleteAll
MsgBox "All links have been deleted. Open the shape's" _
   & Chr(13) & "General property tab to see that there are no links present."
```

{button Links object,JI(`igrafxrf.HLP',`Links_Object')}

Item Method

Syntax *Links*.**Item**(*Index* As Integer) As Link

Description The Item method returns the Link object at the specified *Index* from the Links collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Link. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is helpful to use error trapping before attempting to use the Item method.

Example The following example creates a shape, a diagram, and a link to a diagram. It then uses the Item method to retrieve the new link. The new Link object's Description property is set to a string, and then the link description and target diagram are displayed in a Message Box.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxLinks As Links
Dim igxLink As Link
'Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxLinks variable to a new Links collection object
Set igxLinks = igxShape.Links
' Create a new link and specify which diagram
igxLinks.AddDiagramLink ("NewDiagram")
' Retrieve the last link object
Set igxLink = igxLinks.Item(1)
' Set the Description Property to a string
igxLink.Description = "This is a link to another diagram."
MsgBox "Created a new link" _
    & Chr(13) & "The link is called: " & igxLink.Description _
    & Chr(13) & "The target is called: " & igxLink.Target
```

{button Links object,JI(`igrafxrf.HLP',`Links_Object')}

SubProcessLink Property

Syntax *Links*.**SubProcessLink**

Data Type Link object (read-only, See Object Properties)

Description The SubProcessLink property returns the Link object that is a Shape object's sub-process link. A Shape does not have to have a sub-process link. If one is defined, that Link object is returned. If the Shape does not have a sub-process link, this property returns Nothing.

A sub-process link causes an Entity to jump into a sub-process, return, and then continue it's progress from the location of the link. A shape can have only one sub-process link.

Example The following example sets up a diagram and a sub-process diagram. It then uses the SubProcessLink property to get the link object, and displays it's target diagram.

```
' Dimension the variables
Dim igxDiagramA As Diagram
Dim igxDiagramB As Diagram
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim igxLink As Link
Dim igxSubProcessLink As Link
' Add two diagrams
Set igxDiagramA = ActiveDocument.Diagrams.Add("Diagram A")
igxDiagramA.DiagramObjects.AddTextObject 3000, 500, , , "DIAGRAM A"
Set igxDiagramB = ActiveDocument.Diagrams.Add("Diagram B")
igxDiagramB.DiagramObjects.AddTextObject 3000, 500, , , "DIAGRAM B"
' Add shapes to Diagram A
Set igxShape1 = igxDiagramA.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = igxDiagramA.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector1 = igxDiagramA.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Add shapes to Diagram B
Set igxShape3 = igxDiagramB.DiagramObjects.AddShape(1440, 1440)
Set igxShape4 = igxDiagramB.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxConnector2 = igxDiagramB.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape3, ixDirEast, , , , igxShape4, ixDirWest)
' Add a link from Diagram A to Diagram B
Set igxLink = igxShape2.Links.AddDiagramLink("Diagram B")
' Make the link a sub process
igxLink.IsSubProcess = True
' Add an Entity to the document
ThisDocument.Entities.Add "MyEntity", igxShape1
igxSubProcessLink = igxShape2.Links.SubProcessLink
MsgBox "The sub process links to " & igxSubProcessLink.Target
```

See Also Link object

iGrafx API Object Hierarchy

{button Links object,JI(`igrafxrf.HLP',`Links_Object')}

Note Object

The Note object provides a container for adding textual notes to a shape. In many ways, the Note object is functionally equivalent to a TextBlock object, except that it is a bit more restricted. The Note object is subordinate to the Shape object, and applies only to the Shape object. In fact, there is a one-to-one correspondence between a Shape object and a Note object. Every shape has a note. If the Note is empty, you can still display the Note window. Also, a shape's note can be used and displayed as a field for the shape.

To display the note window, you can do either of the following:

- Select the ViewàNote menu option through the user interface
- Use the Application.ExecuteCommand (ixViewNote) through VBA automation

Properties, Methods, and Events

All of the properties, methods, and events for the Note object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
Paragraphs		
Parent		
TabWidth		
Text		
TextLF		
TextRange		

Related Topics

NoteIndicatorStyle object

iGrafx API Object Hierarchy

Paragraphs Property

Syntax *Note*.**Paragraphs**

Data Type Paragraphs collection object (read-only, See Object Properties)

Description The Paragraphs property returns the Paragraphs collection associated with the specified Note object. The Paragraphs object, through the Item method, provides access to the individual Paragragh objects.

Example The following example creates a shape. It then retrieves the Note object and displays the number of Paragraph objects that are in the note's Paragraphs collection object.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxNote As Note
Dim igxParagraphs As Paragraphs
' Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxNote variable to Note object
Set igxNote = igxShape.Note
' Set the Note object's text
igxNote.Text = "This is a note." & Chr(13) & "It has two paragraphs."
' Set the igxParagraphs variable to the Paragraphs object
Set igxParagraphs = igxNote.Paragraphs
' Display the number of Paragraph objects in the Note object
MsgBox "The number of Paragraphs in the Note object is " & igxParagraphs.Count
```

See Also Paragraph object

Paragraphs object

iGrafx API Object Hierarchy

{button Note object,JI(`igrafxrf.HLP',`Note_Object')}

TabWidth Property

Syntax *Note*.**TabWidth**

Data Type Integer (read/write)

Description The TabWidth property specifies the tab width value for the text in a Note object. The property's value is specified in twips (1440 twips = 1 inch).

Example The following example creates a shape. It then retrieves the Note object and sets the TabWidth property to double the default width.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxNote As Note
' Set igxShape variable to a new Shape object.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxNote variable to Note object.
Set igxNote = igxShape.Note
' Set the Note objects text.
igxNote.Text = "This is a note."
' Set the Note objects TabWidth Property.
igxNote.TabWidth = 100
MsgBox "TabWidth has been set to 100. You can test the TabWidth" _
    & Chr(13) & "by selecting the shape, pressing F6 to view the note, and" _
    & Chr(13) & "pressing the Tab key."
```

{button Note object,JI(`igrafxrf.HLP',`Note_Object')}

TextRange Property

Syntax *Note*.**TextRange**(*First* As Long, *Last* As Long) As TextRange

Data Type TextRange object (read-only, See Object Properties)

Description The TextRange property returns a TextRange object for the specified Note object. The purpose of this property is to provide control over a range of text within a Note.

The TextRange object lets you work with a range of text. The *First* and *Last* arguments specify the start and end positions of the text range. For example, specifying Paragraph1.TextRange(1,5) returns a TextRange that contains the first five characters of the paragraph. Specifying the property without providing the *First* and *Last* arguments returns a TextRange with all the characters in the paragraph. The *First* argument defaults to a value of 1, so to select from the first character of the paragraph only requires specifying the last character.

In addition, each Paragraph object contained within a Note has its own TextRange object that can be used to select either all or part of the paragraph.

Example The following example creates a shape. It then retrieves the shape's Note object and sets it's Text Property. Then the TextRange object is set and it's contents are displayed.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxNote As Note
Dim igxTextRange As TextRange
'Set igxShape variable to a new Shape object.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set igxNote variable to the Note object
Set igxNote = igxShape.Note
' Place text in the Note object
igxNote.Text = "This is a note"
' Set igxTextRange variable to the TextRange object
Set igxTextRange = igxNote.TextRange(1, 6)
' Display the contents of the TextRange object
MsgBox "The contents of the TextRange object is " & igxTextRange.Text
```

See Also TextRange object

iGrafx API Object Hierarchy

{button Note object,JI(`igrafxrf.HLP',`Note_Object')}

ObjectRange Object

The ObjectRange object is a collection of DiagramObject objects that is used to perform various operations on a range of objects. An ObjectRange object is associated with the following objects:

- DiagramObjects object—the ObjectRange contains all objects in the collection.
- Group object—the ObjectRange contains all objects in the group.
- Layer object—the ObjectRange contains all objects on the layer.
- Page object—the ObjectRange contains all objects on the page.

The ObjectRange object associated with each of these objects is automatically populated as the “parent” object is populated. For example, as objects are added to a diagram, the ObjectRange property of the DiagramObjects object contains all of those objects. For the Page object, its ObjectRange property contains all of the DiagramObjects that exist on a specified page.

The ObjectRange object is useful for making Group objects, or for manipulating some set of objects as a set. In addition to the ObjectRange objects associated with the aforementioned objects, you can also make your own ObjectRange variables that can contain any set of objects you specified to be added. To do this, you first have to make an ObjectRange using the MakeObjectRange method through either the Diagram object or the DiagramObject object.

Depending on which ObjectRange object you are dealing with, using certain methods and/or properties makes more sense than others. For example, consider any of the “Add” methods, and in particular the AddAll method. The “Add” methods are useful when you create your own ObjectRange variables—these do not have an association with a specific “parent” object.

However, consider the Page.ObjectRange object, which contains all the objects on a particular page of the diagram. If you use the AddAll method with this, you have just added all the objects in the diagram to the Page’s ObjectRange. This does not affect the page itself; that is, the added objects are not moved to that page. All it means is that you have populated the Page’s ObjectRange with every object, which in most cases would not be desirable since you have then “lost” the inherent association of the page and the objects that reside on the page. The examples provided for the ObjectRange object’s methods and properties attempt to illustrate the intended uses.

Properties, Methods, and Events

All of the properties, methods, and events for the ObjectRange object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Bottom	AddAll	
CenterX	AddByProperty	
CenterY	AddRange	
Count	Align	
DestinationArrowFormat	Angle	
FillFormat	ApplyDefaults	
Height	CenterToPage	
Left	Combine	
LineFormat	ConvertToShape	
Parent	Copy	
Right	Cut	
ShadowFormat	Delete	
SourceArrowFormat	Duplicate	
ThreeDFormat	Flip	
Top	Group	

Width	Item LayerOrder MakeSameSize MoveToLayer Order Remove RemoveAll RemoveByProperty RemoveRange Rotate SnapToGrid SpaceEvenly

Related Topics

Group object

Add Method

Syntax *ObjectRange*.**Add** (*pAddObject* As DiagramObject)

Description The Add method adds one DiagramObject to the specified ObjectRange. The *pAddObject* argument specifies the DiagramObject to add.

Error Specifying an invalid *pAddObject* argument generates an error. Use error trapping if your code could potentially supply an invalid *pAddObject* object name.

Example The following example creates an ObjectRange object, some Shape objects, and then adds the shapes to the ObjectRange.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox "Shapes created.  Now click OK to add them to an ObjectRange."
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
MsgBox "Now click OK to color the shapes blue, at the ObjectRange level."
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Click OK to continue."
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

AddAll Method

Syntax *ObjectRange*.**AddAll**([*optionalType* As IxObjectType])

Description The AddAll method adds all objects in the diagram to the specified ObjectRange. As an option, you can add only those DiagramObject objects of the type specified by the *optionalType* argument.

Using the method makes the most sense for ObjectRange variables that you create with the MakeObjectRange method. Refer to the ObjectRange topic for more information about the use of this method.

The *optionalType* argument limits the type of objects that are added. For instance, if ixObjectShape is specified, the AddAll method adds only shape objects. If you exclude the *optionalType* argument, all objects are added. The IxObjectType constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixObjectShape
1	ixObjectDepartment
3	ixObjectOle
4	ixObjectConnector
5	ixObjectTextGraphic
6	ixObjectGroup
7	ixObjectOther

Example The following example creates an ObjectRange object, a variety of diagram objects, and then adds all the Shape objects to the ObjectRange, using the AddAll(ixObjectShape) method.

```
' Dimension the variables
Dim igxShapeA As Shape
Dim igxShapeB As Shape
Dim igxText As TextGraphicObject
Dim igxObjectRange1 As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShapeA = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShapeB = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
igxShapeA.Text = "Shape A"
igxShapeB.Text = "Shape B"
Set igxText = ActiveDiagram.DiagramObjects.AddTextObject _
    (1440, 1440 * 2, , , "Text Object")
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add the 2 Shapes to the first ObjectRange object
MsgBox "Diagram objects created.  Now click OK to" _
    & Chr(13) & "add all the shape objects to an ObjectRange."
    igxObjectRange1.AddAll (ixObjectShape)
MsgBox "Now click OK to color the shapes blue, at the ObjectRange level."
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Click OK to continue."
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

AddByProperty Method

Syntax *ObjectRange*.**AddByProperty** (*ListName* As String, *PropertyName* As String, *PropertyValue* As Variant)

Description The AddByProperty method adds DiagramObject objects to an ObjectRange based on whether they contain a certain Property and Property value from a specific PropertyList. The AddByProperty method adds a DiagramObject object to the ObjectRange only if it matches all three of the argument values.

The *ListName* argument specifies the name of a PropertyList object. The *PropertyName* argument specifies the name of a Property object. The *PropertyValue* argument specifies the value stored in the property.

Example The following example creates an ObjectRange object, and two shapes with defined Property objects. It then adds shapes to the ObjectRange using the AddByProperty method. Only Shape B is added to the ObjectRange because it contains a Property matching the arguments used with the AddByProperty method.

```
' Dimension the variables
Dim igxShapeA As Shape
Dim igxShapeB As Shape
Dim igxPropertyListA As PropertyList
Dim igxPropertyListB As PropertyList
Dim igxPropertyA As Property
Dim igxPropertyB As Property
Dim igxObjectRange1 As ObjectRange
' Create 2 shapes and assign the shape variables to the 2 Shape objects
Set igxShapeA = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShapeB = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
' Label the shapes
igxShapeA.Text = "Shape A"
igxShapeB.Text = "Shape B"
' Add a PropertyList object to each shape
Set igxPropertyListA = igxShapeA.DiagramObject.PropertyLists.Add("MyList")
Set igxPropertyListB = igxShapeB.DiagramObject.PropertyLists.Add("MyList")
' Add a Property object to each shape
Set igxPropertyA = igxPropertyListA.Add("MyProperty")
Set igxPropertyB = igxPropertyListB.Add("MyProperty")
' Fill each property with a value
igxPropertyA = "1234"
igxPropertyB = "ABCD"
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add a shape to the ObjectRange based on a property
MsgBox "Diagram objects created.  Now click OK to" _
    & Chr(13) & "add shapes to an ObjectRange."
Call igxObjectRange1.AddByProperty("MyList", "MyProperty", "ABCD")
' Change the object in the range to show that it's in the range
MsgBox "Now click OK to color the shapes blue, at the ObjectRange level."
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Shape B was affected because it contained the matching property." _
& Chr(13) & "Click OK to continue."
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

AddRange Method

Syntax *ObjectRange*.**AddRange**(*Range* As ObjectRange)

Description The AddRange method adds DiagramObject objects to the ObjectRange based on the contents of another ObjectRange. The *Range* argument specifies the ObjectRange to add. This method is useful when you want to add a user-defined range into an existing range like the one returned by the Selection property of the Diagram object. The AddRange method redefines the ObjectRange, and does not remove any object that were previously in the ObjectRange.

Example The following example creates two object ranges, adds them together using the AddRange method, and changes the colors of the shapes contained in each Object Range as the contents of the ObjectRanges are changed.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxObjectRange1 As ObjectRange
Dim igxObjectRange2 As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 2)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440 * 2)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
Set igxObjectRange2 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox ("Click OK to define the first ObjectRange and color it blue.")
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.FillFormat.FillColor = vbBlue
' Add the other 2 Shapes to the second ObjectRange object
MsgBox ("Click OK to define the second ObjectRange and color it red.")
igxObjectRange2.Add igxShape3.DiagramObject
igxObjectRange2.Add igxShape4.DiagramObject
igxObjectRange2.FillFormat.FillColor = vbRed
' Add the second ObjectRange to the first ObjectRange
MsgBox "Click OK to combine the two ObjectRanges using the AddRange Method"
igxObjectRange1.AddRange igxObjectRange2
' Change the FillColor of the new ObjectRange to green
MsgBox "Now click OK to change the new ObjectRange to green"
igxObjectRange1.FillFormat.FillColor = vbGreen
' Now change the FillColor of ObjectRange2 back to red
MsgBox "The AddRange Method does not remove the old ObjectRange" & _
Chr(13) & "so click again to change the old ObjectRange back to Red."
igxObjectRange2.FillFormat.FillColor = vbRed
MsgBox "Click OK to continue"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Align Method

Syntax *ObjectRange*.**Align** (*newVal* As IxAlignType)

Description The Align method aligns all of the objects in an ObjectRange in relation to each other. Use the Align method to line up objects into straight rows or columns along their edges or center lines.

The *newVal* argument specifies the type of alignment to perform. The IxAlignType constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixAlignLeft
1	ixAlignHCenter
2	ixAlignRight
3	ixAlignTop
4	ixAlignVCenter
5	ixAlignBottom

Example The following example creates an ObjectRange containing two shapes. The shapes are then aligned along their right sides, their left sides, and then along their vertical centers.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
igxShape2.DiagramObject.Width = 2880
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox ("Click OK to define the ObjectRange.")
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Now click OK to align the ObjectRange to the right edges."
igxObjectRange1.Align (ixAlignRight)
MsgBox "Now click OK to align to the left edges"
igxObjectRange1.Align (ixAlignLeft)
MsgBox "Now click OK to align to the vertical center"
igxObjectRange1.Align (ixAlignVCenter)
MsgBox "Notice the two shapes now overlap, one on top of the other." _
& Chr(13) & "ixAlignVCenter will cause vertically oriented objects to overlap
at their centers." _
& Chr(13) & "Also, ixAlignHCenter will cause horizontally oriented objects to
overlap."
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Angle Method

Syntax *ObjectRange*.**Angle** (*Angle* As Long)

Description The Angle method rotates an ObjectRange object as a unit. The *Angle* argument is in 1/10ths of one degree. For instance, to specify a rotation angle of 90 degrees, the *Angle* argument should be 900. The Angle method is additive. Each time you apply the Angle method, the ObjectRange is rotated further by the amount specified with the *Angle* argument. Rotation occurs about the center of the ObjectRange collection.

Example The following example creates an ObjectRange containing two shapes. It then applies rotation to the ObjectRange twice, using the Angle property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
igxShape2.DiagramObject.Width = 2880
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox ("Click OK to define the ObjectRange.")
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Now click OK to angle the ObjectRange 45 degrees."
igxObjectRange1.Angle (450) ' 45 degrees
MsgBox "Click OK to angle it 135 more degrees."
igxObjectRange1.Angle (1350) '135 degrees
MsgBox "Click OK to continue"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

ApplyDefaults Method

Syntax *ObjectRange*.**ApplyDefaults**

Description The ApplyDefaults method applies the default settings to DiagramObject format properties. This method affects format properties such as FillFormat, ShadowFormat, DestinationArrow format, and others. The effects vary depending on which type of DiagramObjects are contained in the ObjectRange collection.

Example The following example creates an ObjectRange containing two shapes. It changes the fill color of the shapes to blue, and then restores the fill to white using the ApplyDefaults method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
igxShape2.DiagramObject.Width = 2880
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox ("Click OK to define the ObjectRange.")
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
MsgBox "Now click OK to apply a fill color to the ObjectRange."
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Click OK to ApplyDefaults, which will restore the fill color to
white."
igxObjectRange1.ApplyDefaults
MsgBox "Click OK to continue"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Bottom Property

Syntax *ObjectRange*.**Bottom**

Data Type Long (read/write)

Description The Bottom property specifies the distance from the top of the diagram to the bottom of the ObjectRange. The value of the property is specified in twips (1440 twips = 1 inch). The Bottom and CenterY properties both affect the vertical position of the ObjectRange. Changing one changes the other, to reflect the new position.

Example The following example creates an ObjectRange containing two objects. It reads the value of the Bottom property, and moves the bottom of the ObjectRange down by one inch.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 2 shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox "Click OK to define the ObjectRange."
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.FillFormat.FillColor = vbBlue
' Add 1 inch (1440 twips) to the Bottom property
MsgBox "Value of the Bottom property is: " & igxObjectRange.Bottom _
   & Chr(13) & "Click OK to move the bottom down one inch."
igxObjectRange.Bottom = igxObjectRange.Bottom + 1440
' Display the new Bottom value
MsgBox "Value of the Bottom property is: " & igxObjectRange.Bottom _
   & Chr(13) & "Click OK to move the bottom down one inch."
MsgBox "Click OK to continue"
```

See Also CenterY property

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

CenterToPage Method

Syntax *ObjectRange*.**CenterToPage**(*XIndex* As Long, *YIndex* As Long)

Description The CenterToPage method centers the specified ObjectRange object on the specified page. It provides a way to move a range of objects to be centered on a different page, or to center the range of objects on the current page.

The *XIndex* and *YIndex* specify the page on which to center the ObjectRange. Pages are arranged in a grid, and these two arguments specify the location of a page within the grid. For instance, if the diagram has a grid of four pages, the lower right page would be "2, 2".

Note This method does not create new pages. The specified page must already exist before using the CenterToPage method.

.

Example The following example creates an ObjectRange containing two shapes. A third shape is created far off the first page, which creates a page grid of four pages. The ObjectRange is then moved to page 2, 2 (page 4) using the CenterToPage method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange1 As ObjectRange
' Zoom out
ActiveDiagram.Views.Item(1).DiagramView.ZoomPercentage = 50
' Create 2 shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
' Add a shape using large coordinates to put it far off the first page
' This creates a page grid
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(12000, 16000)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox "Click OK to define the ObjectRange."
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
MsgBox "Click OK to move the range to page 2x2."
igxObjectRange1.CenterToPage 2, 2
' Scroll the view to see the result
ActiveDiagram.Views.Item(1).DiagramView.ScrollToPage 4
MsgBox "Click OK to continue"
```

See Also CenterX property

CenterY property

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

CenterX Property

Syntax *ObjectRange*.**CenterX**

Data Type Long (read/write)

Description The CenterX property specifies the distance from the left edge of the diagram to the horizontal center of the ObjectRange. The value of thew property is specified in twips (1440 twips = 1 inch). The Right, Left, and CenterX properties all affect the horizontal position of the ObjectRange. Changing one of these proerties changes all of them, to reflect the new position.

Example The following example creates an ObjectRange containing two objects. It reads the value of the CenterX and CenterY properties, and then adds one inch to the CenterX and CenterY properties.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 2 shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox "Click OK to define the ObjectRange."
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.FillFormat.FillColor = vbBlue
' Add 1 inch (1440 twips) to the CenterX and CenterY properties
MsgBox "Value of the CenterX property is: " & igxObjectRange1.CenterX _
& Chr(13) & "Value of the CenterY property is: " & igxObjectRange1.CenterY _
& Chr(13) & "Click OK to add one inch to the CenterX and CenterY properties."
igxObjectRange1.CenterX = igxObjectRange1.CenterX + 1440
igxObjectRange1.CenterY = igxObjectRange1.CenterY + 1440
MsgBox "Click OK to continue"
```

See Also Left property

Right property

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

CenterY Property

Syntax *ObjectRange*.**CenterY**

Data Type Long (read/write)

Description The CenterY property specifies the distance from the top edge of the diagram to the vertical center of the ObjectRange. The value of the property is specified in twips (1440 twips = 1 inch). The Bottom and CenterY properties both affect the vertical position of the ObjectRange. Changing one changes the other, to reflect the new position.

Example The following example creates an ObjectRange containing two objects. It reads the value of the CenterX and CenterY properties, and then adds one inch to the CenterX and CenterY properties.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 2 shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
' Set the igxObjectRange variable to the ObjectRange object.
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox "Click OK to define the ObjectRange."
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.FillFormat.FillColor = vbBlue
' Add 1 inch to the CenterX and CenterY properties
MsgBox "Value of the CenterX property is: " & igxObjectRange1.CenterX _
& Chr(13) & "Value of the CenterY property is: " & igxObjectRange1.CenterY _
& Chr(13) & "Click OK to add one inch to the CenterX and CenterY properties."
igxObjectRange1.CenterX = igxObjectRange1.CenterX + 1440
igxObjectRange1.CenterY = igxObjectRange1.CenterY + 1440
MsgBox "Click OK to continue"
```

See Also Bottom property

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Combine Method

Syntax *ObjectRange*.**Combine**(*CombineType* As IxCombineType)

Description The Combine method combines all TextGraphicObject objects in the specified ObjectRange according to the rules of the method specified by the *CombineType* argument. The CombineType values are equivalent to the Combine methods found in the Arrange->Combine menu.

The Combine method works only with TextGraphicObject objects that contain a graphic. It does not work with Shapes, Legends, ConnectorLines, OLEObjects, or Bitmaps. The TextGraphicObject objects also can contain text; however, once the Combine operation is performed, the text in any of the object involved is removed.

Note that the order that the objects are added to the ObjectRange is important, depending on the Combine method you plan to use. The results of various methods are NOT the same if the order of the objects in the range differs. It is useful to experiment through the user interface with the various Combine methods, and the order in which the objects are selected.

The *CombineType* argument. The IxCombineType constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixConnectOpen
1	ixConnectClosed
2	ixDisconnect
3	ixIntersect
4	ixSlice
5	ixUnion
6	ixMakeOneShape
7	ixMakeOneGraphic

Example The following example creates an overlapping rectangle and ellipse, and adds them to an ObjectRange. Then the Combine method is used to Intersect the graphics.

```
' Dimension the variables
Dim igxBuilder1 As New GraphicBuilder
Dim igxBuilder2 As New GraphicBuilder
Dim igxTGObj As TextGraphicObject
Dim igxObjectRange As ObjectRange
' Build a rectangle and ellipse
igxBuilder1.Rectangle 0, 0, 1, 1
igxBuilder2.Ellipse 0, 0, 1, 1
' Add the rectangle and ellipse to the diagram
Set igxTGObj = ActiveDiagram.DiagramObjects.AddGraphic _
    (igxBuilder1.Graphic, 1440 * 3, 1440 * 2, 3000, 2000)
igxTGObj.Text = "Graphic One"
Set igxTGObj = ActiveDiagram.DiagramObjects.AddGraphic _
    (igxBuilder2.Graphic, 1440 * 4, 1440 * 1.5, 2000, 2600)
igxTGObj.Text = "Graphic Two"
MsgBox "Added two TextGraphicObjects to the diagram."
' Create a new ObjectRange
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add the graphics to the range
```

```
MsgBox "Click OK to add the graphics to the ObjectRange"
igxObjectRange.AddAll ixObjectTextGraphic
' Perform the Combine Intersect method
MsgBox "Click OK to Combine/Outline the graphics"
igxObjectRange.Combine ixOutline
MsgBox "Click OK to continue. Note that the text " _
    & "has been removed."
```

See Also TextGraphicObject object

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

ConvertToShape Method

Syntax *ObjectRange*.**ConvertToShape**

Description The ConvertToShape method combines all the TextGraphicObject objects in the ObjectRange into a single Shape object. If any of the objects in the range also contain text, only the text of the first object in the range is kept; text from all other objects in the range is discarded

The new Shape object is positioned at the upper left most corner of the ObjectRange. The size of the new Shape is determined by the largest vertical size among the Graphic objects, and the largest hoizontal size among the Graphic objects.

Each Graphic becomes an Item in the new Shape's *Shape.Graphic.GraphicGroup.Graphics* collection. Each Graphic is drawn inside the new shape according to the Graphic's original definition.

The ConvertToShape method converts only the TextGraphicObject objects in the ObjectRange. Other types of diagram objects, including Shapes, Legends, ConnectorLines, and OLE Objects, remain in the ObjectRange unaltered.

It is best to use this method with an ObjectRange that you have created using the MakeObjectRange method.

Example The following example creates two Graphic objects, and adds them to an ObjectRange. Then the ConvertToShape method is applied, which combines the Graphic objects into a single new Shape object.

```
' Dimension the variables
Dim igxBuilder1 As New GraphicBuilder
Dim igxBuilder2 As New GraphicBuilder
Dim igxTGObj As TextGraphicObject
Dim igxObjectRange As ObjectRange
' Create a new ObjectRange
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Build a rectangle and ellipse
igxBuilder1.Rectangle 0, 0, 1, 1
igxBuilder2.Ellipse 0, 0, 1, 1
' Add the rectangle and ellipse to the diagram
Set igxTGObj = ActiveDiagram.DiagramObjects.AddGraphic _
    (igxBuilder1.Graphic, 1440 * 3, 1440 * 2, 3000, 2000)
igxTGObj.Text = "Graphic One"
' Add the graphic to the range
MsgBox "Click OK to add the graphic to the ObjectRange"
igxObjectRange.Add igxTGObj.DiagramObject
Set igxTGObj = ActiveDiagram.DiagramObjects.AddGraphic _
    (igxBuilder2.Graphic, 1440 * 4, 1440 * 1.5, 2000, 2600)
igxTGObj.Text = "Graphic Two"
' Add the graphic to the range
MsgBox "Click OK to add the graphic to the ObjectRange"
igxObjectRange.Add igxTGObj.DiagramObject
MsgBox "Added two TextGraphicObjects to the diagram."
' Convert the TextGraphicObjects in the range into a shape
MsgBox "Click OK to convert the graphic objects to a Shape."
igxObjectRange.ConvertToShape
MsgBox "Click OK to continue. Note that only the text " _
    & "from the first object added to the range has been kept."
```

See Also TextGraphicObject object

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Copy Method

Syntax *ObjectRange*.**Copy**

Description The Copy method copies all of the objects in the ObjectRange to the clipboard. This is equivalent to using the Edit->Copy menu item.

Example The following example creates an ObjectRange containing two objects. It then copies the objects to the clipboard, deletes the ObjectRange from the diagram, and then restores the objects using the Diagram.Paste method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 2 shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(2880, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(2880, 2880)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox "Click OK to define the ObjectRange."
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Click OK to copy the range"
igxObjectRange1.Copy
MsgBox "Click OK to delete the range"
igxObjectRange1.Delete
MsgBox "Click OK to paste the range"
ActiveDiagram.Paste 1000, 1000
MsgBox "Click OK to continue"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Cut Method

Syntax *ObjectRange*.**Cut**

Description The Cut method cuts all of the objects in the ObjectRange to the clipboard. This is equivalent to the Edit->Cut menu item.

Example The following example creates an ObjectRange containing two objects. It then copies the objects to the clipboard, deletes the ObjectRange, and restores the objects using the Diagram.Paste method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 2 shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(2880, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(2880, 2880)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the first ObjectRange object
MsgBox "Click OK to define the ObjectRange."
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Click OK to Cut the range"
igxObjectRange1.Cut
MsgBox "Click OK to paste the range"
ActiveDiagram.Paste 1000, 1000
MsgBox "Click OK to continue"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

DestinationArrowFormat Property

Syntax *ObjectRange*.**DestinationArrowFormat**

Data Type ArrowFormat object (read-only, See Object Properties)

Description The DestinationArrowFormat property returns an ArrowFormat object for the specified ObjectRange object. Use the DestinationArrowFormat property to copy arrow styles from one object to another, or to access the destination arrow format properties within an ObjectRange, such as Color, Size, and Style properties. The DestinationArrowFormat property returns values for the Color, Size, and Style properties, but if the ObjectRange contains more than one destination arrow, only matching properties will have meaningful return values. Non-matching destination arrow properties return a value of -1.

Example The following example creates an ObjectRange containing three shapes and two connector lines. The connector line destination arrows are different colors. It then sets all the arrows to the same color using the DestinationArrowFormat property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxConnLine2 As ConnectorLine
Dim igxObjectRange1 As ObjectRange
Dim igxArrowFormat1 As ArrowFormat
' Create shapes In the active diagram
Set igxShape1 = ActiveDiagram.DiagramObjects. _
   AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects. _
   AddShape(1440 * 4, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects. _
   AddShape(1440 * 7, 1440)
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = Application.ActiveDiagram.DiagramObjects. _
   AddConnectorLine(RouteType:=ixRouteRightAngle, _
   RouteFlag:=ixRouteFlagFindEdge, SourceShape:=igxShape1, _
   SourceDir:=ixDirEast, SourceConnectType:=ixConnectRelativeToShape, _
   DestShape:=igxShape2, DestDir:=ixDirWest, _
   DestConnectType:=ixConnectRelativeToShape)
' Get the ConnectorFormat object
With igxConnLine1.ConnectorFormat
   .DestinationArrowFormat.Color = vbBlue
   .DestinationArrowFormat.Size = 3
   .LineFormat.Color = vbRed
   .LineFormat.Width = 3
   .RepeatDestinationArrow = True
End With
' Draw a connector line between shapes 2 and 3
Set igxConnLine2 = Application.ActiveDiagram.DiagramObjects. _
   AddConnectorLine(RouteType:=ixRouteRightAngle, _
   RouteFlag:=ixRouteFlagFindEdge, SourceShape:=igxShape2, _
   SourceDir:=ixDirEast, SourceConnectType:=ixConnectRelativeToShape, _
   DestShape:=igxShape3, DestDir:=ixDirWest, _
   DestConnectType:=ixConnectRelativeToShape)
' Get the ConnectorFormat object
```

```
With igxConnLine2.ConnectorFormat
    .DestinationArrowFormat.Color = vbRed
    .DestinationArrowFormat.Size = 3
    .LineFormat.Color = vbRed
    .LineFormat.Width = 3
    .RepeatDestinationArrow = True
End With
' Create the ObjectRange objects
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add the shapes and connector lines to the range
Call igxObjectRange1.Add(igxShape1.DiagramObject)
Call igxObjectRange1.Add(igxShape2.DiagramObject)
Call igxObjectRange1.Add(igxConnLine1.DiagramObject)
Call igxObjectRange1.Add(igxConnLine2.DiagramObject)
' Set the first ObjectRange's arrow format
Set igxArrowFormat1 = igxObjectRange1.DestinationArrowFormat
' Use DestinationArrowFormat to make all the arrows black
MsgBox "Click OK to make all the arrows black"
igxObjectRange1.DestinationArrowFormat.Color = vbBlack
MsgBox "Click OK to continue."
```

See Also

ArrowFormat object

iGrafx API Object Hierarchy

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Duplicate Method

Syntax *ObjectRange*.**Duplicate**(*XOffset* As Long, *YOffset* As Long)

Description The Duplicate method duplicates all of the objects in the ObjectRange, creating new DiagramObject objects. It does not create a new ObjectRange object, nor are the duplicated objects added to the ObjectRange.

The new duplicate objects are created in the same position as the source objects, and then the source ObjectRange is moved by the amount specified with *XOffset* and *YOffset* arguments. The *XOffset* and *YOffset* arguments specify a location to move the source ObjectRange, relative to the position of the ObjectRange before duplicating it. If you specify 0 for both the arguments, the source objects overlay the duplicates.

The new duplicate objects inherit the index numbers of the source objects in the DiagramObjects collection. The source objects are then indexed after the new duplicates in the DiagramObjects collection.That is, if you duplicate items 1 and 2 out of a total of 3 objects in the DiagramObjects collection, then the duplicates would be items 1 and 2, and the source objects would be items 4 and 5.

Example The following example creates an ObjectRange containing three shapes. It adds the first and third shapes to an object range, and then duplicates those objects using the Duplicate method. The original objects are moved 3.5 inches to the right, using the XOffset and YOffset arguments. The original objects in the object range then have their fill color set to blue. Finally, each object is selected from the DiagramObjects collection to show its position in the collection. As specified, the duplicated objects take the position in the DiagramObjects collection of the original objects. The original objects are appended to the end of the collection.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange1 As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 3)
MsgBox "View the diagram"
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add two of the Shapes to the ObjectRange object
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape3.DiagramObject
MsgBox "Added two shapes to an object range. Click OK " _
    & Chr(13) & "to duplicate the shapes in the object range."
' Duplicate the current ObjectRange
igxObjectRange1.Duplicate 1440 * 3.5, 0
' Set a blue fill for the original objects
MsgBox "Click OK to set a blue fill for the Source objects"
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "The source objects were moved by the " _
    & "offset amount, not the duplicates."
' Select each object and display its position in the
' DiagramObjects collection
For iCount = 1 To ActiveDiagram.DiagramObjects.Count
    ActiveDiagram.DiagramObjects.Item(iCount).Selected = True
    MsgBox "The selected object is Item " & iCount _
```

```
            & " in the DiagramObjects collection."
        ActiveDiagram.DiagramObjects.Item(iCount).Selected = False
    Next iCount
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

FillFormat Property

Syntax *ObjectRange*.**FillFormat**

Data Type FillFormat object (read-only, See Object Properties)

Description The FillFormat property returns the FillFormat object for the specified ObjectRange object. The FillFormat object controls whether a fill is used, and if so, what type of fill (solid, pattern, or gradient), and the color or colors used.

Example The following example creates three shapes and selects two of the shapes to assign to an ObjectRange object. The FillFormat property is then used to change the color of the shapes in the ObjectRange object to blue.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Set the FillColor for the shapes in the ObjectRange to blue
MsgBox ("Click OK to set a blue fill for the object range")
igxObjectRange.FillFormat.FillColor = vbBlue
MsgBox ("Click OK to continue")
```

See Also FillFormat object

iGrafx API Object Hierarchy

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Flip Method

Syntax *ObjectRange*.**Flip** (*FlipType* As IxFlipType)

Description The Flip method flips each object contained in the ObjectRange in the direction specified by the *FlipType* argument. The Flip method flips each object in the range separately; it does not flip the entire ObjectRange as a single unit. All text, field text and custom data being displayed, and shape numbers are ignored.

The *FlipType* argument specifies which direction, horizontal or vertical, to flip the diagram objects. The IxFlipType constant defines the valid values, which are listed in the following table.

Value	Name of Constant	Description
0	ixFlipHorizontal	Flips objects right to left
1	ixFlipVertical	Flips objects top to bottom

Example The following example creates an ObjectRange containing three shapes. It then flips each of the objects in the ObjectRange using the Flip method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShapeItem As ShapeLibraryItem
Dim igxObjectRange As ObjectRange
' Set a shape item variable as a parallelogram. Boxes, circles, etc…
' don’t change appearance when they are flipped
Set igxShapeItem = Application.ShapeLibraries(1).Item(1)
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440, igxShapeItem)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 2, 1440 * 2, igxShapeItem)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440, igxShapeItem)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Set the FillColor for the shapes in the ObjectRange to blue
MsgBox "Click OK to invoke the Flip method."
igxObjectRange.Flip (ixFlipVertical)
MsgBox "Click OK to continue"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Group Method

Syntax *ObjectRange*.**Group** As Group

Description The Group method makes a group (a Group object) out of all of the objects in the ObjectRange. If you delete a DiagramObject from the source ObjectRange after making a group, the remaining objects are ungrouped. If you remove a DiagramObject from the ObjectRange, the Group remains unaltered. The method returns a new Group object, and the result must be assigned to a variable of type Group.

Example The following example creates an ObjectRange containing three shape objects. It then creates a group from the ObjectRange. Finally it displays the relationship between the Group object and the ObjectRange object.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
Dim igxGroup As Group
Dim String1 As String
Dim String2 As String
Dim Index As Integer
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3.5, 1440)
igxShape1.DiagramObject.ObjectName = "ShapeA"
igxShape2.DiagramObject.ObjectName = "ShapeB"
igxShape3.DiagramObject.ObjectName = "ShapeC"
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Group all the objects in the ObjectRange
MsgBox "Click OK to group the objects"
Set igxGroup = igxObjectRange.Group
' Build strings containing the ObjectName of each object
String1 = ""
String2 = ""
' ObjectRange
For Index = 1 To igxObjectRange.Count
    String1 = String1 + igxObjectRange.Item(Index).ObjectName + Chr(13)
Next Index
' Group
For Index = 1 To igxGroup.ObjectRange.Count
    String2 = String2 + igxGroup.ObjectRange.Item(Index).ObjectName + Chr(13)
Next Index
MsgBox "The ObjectRange contains these shapes:" & Chr(13) & String1 _
    & Chr(13) & _
   "The Group contains these shapes:" & Chr(13) & String2
```

See Also Group object

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Item Method

Syntax *ObjectRange*.**Item**(*Index* As Integer) As DiagramObject

Description The Item method returns the DiagramObject object at the specified *Index* from the specified ObjectRange collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type DiagramObject. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. Use error trapping to handle these errors.

Example The following example iterates through the diagram objects in an ObjectRange, and changes the size of each object.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Iterate through the objects and resize them
For Index = 1 To igxObjectRange.Count
    MsgBox "Click OK to move to resize Item(" & Index & ")"
    Call igxObjectRange.Item(Index).Resize(2000, 2000)
Next Index
MsgBox "Click OK to continue"
```

LayerOrder Method

Syntax *ObjectRange*.**LayerOrder** *LayerOrderType* As IxLayerOrderType

Description The LayerOrder method moves the objects in an ObjectRange up or down one layer. It is possible for an ObjectRange to contain objects from different layers. The LayerOrder method preserves the relative position of object in layers unless there is no layer to which to move an object. If there is no layer for an object to move up or down to, then that object stays on it's current layer.

The *LayerOrderType* argument specifies the direction to move objects. The IxLayerOrderType constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixMoveUp
1	ixMoveDown

Example The following example creates an ObjectRange containing three shapes. It adds a new layer to the diagram, and then moves the ObjectRange diagram objects up to the new layer.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
Dim iLayersCount As Integer
Dim iLayerIndex As Integer
' If there are any above Layer 1, delete them
iLayersCount = ActiveDiagram.Layers.Count
If iLayersCount > 1 Then
    For iLayerIndex = 2 To iLayersCount
        ' Deleting Item 2 each time, because each Delete
        ' shifts the Item Indexes
        ActiveDiagram.Layers.Item(2).Delete
    Next iLayerIndex
End If
' Create a new layer and activate Layer 1
ActiveDiagram.Layers.Add ("Layer 2")
ActiveDiagram.Layers.Item(1).Activate
' Create 3 shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Move the ObjectRange up one layer
MsgBox "Click OK to move the ObjectRange up to Layer 2"
igxObjectRange.LayerOrder (ixMoveUp)
MsgBox "The items in the ObjectRange have been moved up to Layer 2"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Left Property

Syntax *ObjectRange*.**Left**

Data Type Long (read/write)

Description The Left property specifies the location of the left side of the ObjectRange object. The units are specified in twips (1440 twips = 1 inch). The Right, Left, and CenterX properties all affect the horizontal position of the ObjectRange. Changing one of these proerties changes all of them, to reflect the new position.

Example The following example moves the ObjectRange so that it's left edge is 3 inches from the left side of the diagram.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Use the Left property to reposition the ObjectRange
MsgBox "Currently the ObjectRange Left position is " & igxObjectRange.Left _
   & Chr(13) & "and the ObjectRange Right position is " _
   & igxObjectRange.Right & Chr(13) & _
   "Click OK to move the ObjectRange by changing the Left property."
igxObjectRange.Left = 1440 * 3
MsgBox "Now the ObjectRange Left position is " & igxObjectRange.Left & "." _
   & Chr(13) & "and the ObjectRange Right position is " & igxObjectRange.Right
```

See Also CenterX property

Right property

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

LineFormat Property

Syntax *ObjectRange*.**LineFormat**

Data Type LineFormat object (read-only, See Object Properties)

Description The LineFormat property returns a LineFormat object for the specified ObjectRange object. This property allows you to change all of the line formatting attributes of the objects in the ObjectRange that use lines, such as shapes and connector lines.

Example The following example copies the LineFormat object from one ObjectRange to another, so that the second ObjectRange takes on the LineFormat properties of the first ObjectRange.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxObjectRange1 As ObjectRange
Dim igxObjectRange2 As ObjectRange
' Create shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440 * 2)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
Set igxObjectRange2 = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange objects
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange2.Add igxShape3.DiagramObject
igxObjectRange2.Add igxShape4.DiagramObject
' Give ObjectRange2 a custom LineFormat
igxObjectRange2.LineFormat.Width = 100
MsgBox "ObjectRange1's LineFormat is set. Click OK to copy ObjectRange1's" _
    & Chr(13) & "LineFormat object to ObjectRange2's LineFormat object"
' Copy the LineFormat object from one ObjectRange to another
igxObjectRange2.LineFormat = igxObjectRange1.LineFormat
MsgBox "Click OK to continue"
```

See Also LineFormat object

iGrafx API Object Hierarchy

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

MakeSameSize Method

Syntax *ObjectRange*.**MakeSameSize** (*newVal* As IxAccordingTo)

Description The MakeSameSize method makes all of the objects in the ObjectRange the same size according to attributes specified by the *newVal* argument.

The *newVal* argument specifies which attributes of objects are made the same size. The IxAccordingTo constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixSizeWidth
1	ixSizeHeight
2	ixSizeBoth
3	ixSizeTextFit

Example The following example creates an ObjectRange containing two shapes of different sizes. It then uses the MakeSameSize method to resize the shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxObjectRange1 As ObjectRange
Dim igxObjectRange2 As ObjectRange
' Create shapes and assign the shape variables to the Shape objects.
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 2)
igxShape1.DiagramObject.Width = 3000
igxShape1.Text = "This is an activity in the process with a long text label"
igxShape2.Text = "Activity B"
' Set the igxObjectRange variable to the ObjectRange object.
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange objects.
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
MsgBox "do it"
' Copy the LineFormat object from one ObjectRange to another
igxObjectRange1.MakeSameSize (ixSizeTextFit)
MsgBox "Click OK to continue"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

MoveToLayer Method

Syntax *ObjectRange*.**MoveToLayer** *LayerIndex* As Long

Description The MoveToLayer method moves objects contained in an ObjectRange to a specific layer. It is possible for an ObjectRange to contain objects that are on different layers. Use the MoveToLayer method to move all the objects to the same layer. The *LayerIndex* argument specifies the destination layer for the objects.

Error Specifying an invalid *LayerIndex* value generates an error. Use error trapping if your code could potentially specify a *LayerIndex* for a Layer that does not exist.

Example The following example creates an ObjectRange containing objects that reside on Layers 1 and 2. It then moves all the objects to Layer 3 using the MoveToLayer method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange1 As ObjectRange
Dim iLayersCount As Integer
Dim iLayerIndex As Integer
' If there are any above Layer 1, delete them
iLayersCount = ActiveDiagram.Layers.Count
If iLayersCount > 1 Then
    For iLayerIndex = 2 To iLayersCount
        ' Deleting Item 2 each time, because each Delete
        ' shifts the Item Indexes
        ActiveDiagram.Layers.Item(2).Delete
    Next iLayerIndex
End If
' Create a new layer and activate Layer 1
ActiveDiagram.Layers.Add ("Layer 2")
ActiveDiagram.Layers.Add ("Layer 3")
ActiveDiagram.Layers.Item(1).Activate
' Create 3 shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 2)
igxShape1.Text = "This shape on Layer 1"
igxShape2.Text = "This shape on Layer 1"
' Activate Layer 2 and put the third shape there
ActiveDiagram.Layers.Item(2).Activate
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
igxShape3.Text = "This shape on Layer 2"
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add the Shapes to the ObjectRange object
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.Add igxShape3.DiagramObject
' Move the ObjectRange up one layer
MsgBox "Click OK to move all the objects to Layer 3 " _
    & "using the MoveToLayer method."
igxObjectRange1.MoveToLayer (3)
igxShape1.Text = "This shape on Layer 3"
igxShape2.Text = "This shape on Layer 3"
```

```
igxShape3.Text = "This shape on Layer 3"
MsgBox "The items in the ObjectRange have been moved to Layer 3"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Order Method

Syntax *ObjectRange*.**Order** (*OrderType* As IxOrderType)

Description The Order method changes the display order of all the objects in the ObjectRange. The display order determines the order in which diagram objects are drawn. If objects overlap, objects drawn first are covered by objects drawn later.

The *OrderType* argument specifies how objects in a range are ordered (front or back) when they overlap each other. The IxOrderType constant defines the valid values, which are listed in the following table.

Value	Name of Constant
0	ixBringForward
1	ixBringToFront
2	ixSendBackward
3	ixSendToBack

Example The following example creates four shapes, the last two overlapping the first two. The last two shapes are added to an ObjectRange. Then the Order method is used to send the last two shapes to the back, reversing the display order.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxObjectRange1 As ObjectRange
' Create shapes and assign the shape variables to the Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 + 720)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 * 3, 1440 * 3 - 720)
' Color the last 2 shapes blue
igxShape3.FillColor = vbBlue
igxShape4.FillColor = vbBlue
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add the last 2 shapes to the ObjectRange object
igxObjectRange1.Add igxShape3.DiagramObject
igxObjectRange1.Add igxShape4.DiagramObject
' SendToBack the contents of the ObjectRange
MsgBox "Click OK to send the ObjectRange members to the back."
igxObjectRange1.Order (ixSendToBack)
MsgBox "Click OK to continue"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Remove Method

Syntax *ObjectRange*.**Remove** (*pRemoveObject* As DiagramObject)

Description The Remove method removes one DiagramObject from the specified ObjectRange. The *pRemoveObject* argument specifies the DiagramObject to remove.

Error Specifying an invalid *pRemoveObject* argument generates an error. Use error trapping if your code could potentially supply an invalid *pRemoveObject* object name.

Example The following example creates an ObjectRange containing three shapes. It uses the Remove method to remove one of the shapes from the object range.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
MsgBox "Click OK to apply a Blue fill color to the current ObjectRange"
' Set the FillColor to blue for the shapes
igxObjectRange.FillFormat.FillColor = vbBlue
' Remove one of the shapes from the ObjectRange
MsgBox "Click again to remove one of the shapes from the ObjectRange" _
    & Chr(13) & "using ObjectRange.Remove and fill the remaining " _
    & "shape with Green."
igxObjectRange.Remove igxShape1.DiagramObject
' Set the FillColor to green for the 1 shape remaining
igxObjectRange.FillFormat.FillColor = vbGreen
MsgBox "Click OK to continue."
```

RemoveAll Method

Syntax *ObjectRange*.**RemoveAll**([*optionalType* As IxObjectType])

Description The RemoveAll method removes all objects from the specified ObjectRange. As an option, you can limit the removal to only those DiagramObject objects of the type specified by the *optionalType* argument.

The *optionalType* argument limits the type of objects that are removed. For instance, if ixObjectShape is specified, the RemoveAll method removes only shape objects. If you exclude the *optionalType* argument, all objects are removed. The IxObjectType constant defines the valid values, and are listed in the following table.

Value	Name of Constant
0	ixObjectShape
1	ixObjectDepartment
3	ixObjectOle
4	ixObjectConnector
5	ixObjectTextGraphic
6	ixObjectGroup
7	ixObjectOther

Example The following example creates an ObjectRange containing three shapes. The shapes in the ObjectRange are changed to blue, and then the shapes are removed from the ObjectRange object using the RemoveAll method with the *optionalType* argument set to ixObjectShape. Finally, an attempt is made to set ObjectRange fill color to red, which has no effect because all shapes have been removed.

```
' Dimension variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
MsgBox "Click OK to apply a fill color to the current ObjectRange"
' Set the FillColor to blue for the shapes
igxObjectRange.FillFormat.FillColor = vbBlue
' Removes all the shapes from the ObjectRange
MsgBox "Click again to invoke the RemoveAll(ixObjectShape) method"
igxObjectRange.RemoveAll (ixObjectShape)
MsgBox "Now click again to apply a Red fill color"
igxObjectRange.FillFormat.FillColor = vbRed
MsgBox "Had no effect because RemoveAll left no shapes in the ObjectRange."
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

RemoveByProperty Method

Syntax *ObjectRange*.**RemoveByProperty**(*ListName* As String, *PropertyName* As String, *PropertyValue* As Variant)

Description The RemoveByProperty method removes DiagramObject objects from the specified ObjectRange based on a Property object. The three arguments specify the property to locate within the object range. If a DiagramObject has a matching property, it is removed from the object range. All three argument values must match exactly.

The *ListName* argument specifies the name of a PropertyList object. The *PropertyName* argument specifies the name of a Property object. The *PropertyValue* argument specifies the value stored in the property designated by *PropertyName*.

Example The following example creates an ObjectRange containing two shapes. Each shape is given a Property called Cost. One shape has its Cost value set to zero, the other is set to $231. It then removes the shape with a cost of $0.00, using the RemoveByProperty method.

```
' Dimension the variables
Dim igxShapeA As Shape
Dim igxShapeB As Shape
Dim igxPropertyListA As PropertyList
Dim igxPropertyListB As PropertyList
Dim igxCostOfA As Property
Dim igxCostOfB As Property
Dim igxObjectRange1 As ObjectRange
' Create 2 shapes and assign the shape variables to the 2 Shape objects.
Set igxShapeA = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440)
Set igxShapeB = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
' Label the shapes
igxShapeA.Text = "Activity A" + Chr(13) + "Cost $0.00"
igxShapeB.Text = "Activity B" + Chr(13) + "Cost $231.00"
' Add a PropertyList object to each shape
Set igxPropertyListA = igxShapeA.DiagramObject.PropertyLists.Add("MyList")
Set igxPropertyListB = igxShapeB.DiagramObject.PropertyLists.Add("MyList")
' Add a Property object to each shape
Set igxCostOfA = igxPropertyListA.Add("Cost")
Set igxCostOfB = igxPropertyListB.Add("Cost")
' Specify a cost for each activity
igxCostOfA = 0
igxCostOfB = 231
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add the shapes to the ObjectRange
Call igxObjectRange1.AddAll(ixObjectShape)
MsgBox "Shapes created and added to the ObjectRange. Each " _
    & "shape has a property called Cost." _
    & Chr(13) & "The Cost Property of Activity A has been " _
    & "set to $  0.00" & Chr(13) & "The Cost Property of " _
    & "Activity B has been set to $231.00"
' Remove any DiagramObjects from the ObjectRange with a
' Cost property = 0
MsgBox "Click OK to remove from the ObjectRange any shape " _
    & "with a Cost of $0.00"
Call igxObjectRange1.RemoveByProperty("MyList", "Cost", 0)
' Apply a fill color, which will not affect removed objects
```

```
MsgBox "Now apply a fill color to the ObjectRange."
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Activity A was not affected because it was removed " _
    & "from the ObjectRange."
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

RemoveRange Method

Syntax *ObjectRange*.**RemoveRange**(*Range* As ObjectRange)

Description The RemoveRange method removes DiagramObject objects from tha specified ObjectRange based on the contents of another ObjectRange, designated by the *Range* argument. The *Range* argument specifies the name of an ObjectRange which may have diagram objects in common with the specified ObjectRange. If the two ObjectRange objects have any DiagramObjects in common, those DiagramObjects are removed from the specified ObjectRange. If the two ObjectRanges have no DiagramObjects in common, the RemoveRange method has no effect.

Error Specifying an invalid *Range* argument generates an error. Use error trapping if your code could potentially supply to the *Range* argument an ObjectRange name that doesn't exist.

Example The following example creates two ObjectRanges and several shapes. The two object ranges have several shapes in common. Shapes are then removed from the first ObjectRange based on the contents of the ObjectRange given as the *Range* argument.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxShape4 As Shape
Dim igxObjectRange1 As ObjectRange
Dim igxObjectRange2 As ObjectRange
' Create 4 shapes and assign the shape variables to the 4 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440)
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 4)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
Set igxObjectRange2 = ActiveDiagram.MakeObjectRange
' Add 3 of the Shapes to ObjectRange1
igxObjectRange1.Add igxShape1.DiagramObject
igxObjectRange1.Add igxShape2.DiagramObject
igxObjectRange1.Add igxShape3.DiagramObject
' Add 3 of the shapes to ObjectRange2
igxObjectRange2.Add igxShape2.DiagramObject
igxObjectRange2.Add igxShape3.DiagramObject
igxObjectRange2.Add igxShape4.DiagramObject
' Our two ObjectRanges overlap. Shapes 2 and 3
' are in both ObjectRanges.
' Show which shapes are in which ObjectRanges by highlighting
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Object in blue are members of the first ObjectRange"
igxObjectRange1.FillFormat.FillColor = vbWhite
igxObjectRange2.FillFormat.FillColor = vbBlue
MsgBox "Object in blue are members of the second ObjectRange"
' Remove shapes from ObjectRange1 using the RemoveRange method
MsgBox "Now click OK to remove diagram objects from the first ObjectRange" _
& Chr(13) & "based on the contents of the second ObjectRange."
igxObjectRange1.RemoveRange igxObjectRange2
' Now display the result of the RemoveRange method
igxShape1.FillColor = vbWhite
igxShape2.FillColor = vbWhite
```

```
igxShape3.FillColor = vbWhite
igxShape4.FillColor = vbWhite
igxObjectRange1.FillFormat.FillColor = vbBlue
MsgBox "Object in blue are members of the first ObjectRange"
igxObjectRange1.FillFormat.FillColor = vbWhite
igxObjectRange2.FillFormat.FillColor = vbBlue
MsgBox "Object in blue are members of the second ObjectRange"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Right Property

Syntax *ObjectRange*.**Right**

Data Type Long (read/write)

Description The Right property specifies the location of the right side of the ObjectRange object. The units for this property are specified in twips (1440 twips = 1 inch). The Right, Left, and CenterX properties all affect the horizontal position of the ObjectRange. Changing one of these properties changes all of them, to reflect the new position.

Example The following example creates three shapes and selects two of the shapes to be assigned to an ObjectRange object. The Right Property is then used to adjust the position of the right boundary of the ObjectRange.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
MsgBox ("Click OK to change the Right property.")
' Set the FillColor to blue for the shapes
igxObjectRange.FillFormat.FillColor = vbBlue
' Sets the right side of the object range at 7880 twips
igxObjectRange.Right = 7880
MsgBox ("Right side of object range moved to 7880")
```

See Also CenterX property

Left property

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

Rotate Method

Syntax *ObjectRange*.**Rotate** (*RotateType* As IxRotateType)

Description The Rotate method rotates each shape in the ObjectRange 90 degrees. Rotation is either to the left or to the right. Each diagram object in the ObjectRange is rotated individually. The Rotate method does not rotate the entire collection as one unit.

The *RotateType* argument specifies which direction to rotate the diagram objects. The IxRotateType constant defines the valid values, and are listed in the following table.

Value	Name of Constant
0	ixRotateLeft
1	ixRotateRight

Example The following example creates three shapes and selects two of the shapes to be assigned to an ObjectRange object. The Rotate method is then used to rotate the ObjectRange shapes.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Set the FillColor to blue for the shapes
MsgBox ("Click OK to rotate the shapes and set the fill to blue.")
igxObjectRange.FillFormat.FillColor = vbBlue
' Rotate the shapes in the ObjectRange
igxObjectRange.Rotate (ixRotateRight)
MsgBox ("Shapes filled and rotated.")
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

ShadowFormat Property

Syntax *ObjectRange*.**ShadowFormat**

Data Type ShadowFormat object (read-only, See Object Properties)

Description The ShadowFormat property returns a ShadowFormat object. This object is used to define the shadow formatting characteristics for the object range. The formatting hierarchy for shapes allows the developer to customize and manipulate shapes individually.

Example The following example creates three shapes and selects two of the shapes to be assigned to an ObjectRange object. The ShadowFormat property is then used to change the shadow formatting of the shapes in the object range.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Set the ObjectRange's ShadowFormat property
igxObjectRange.ShadowFormat.Type = ixShadow14
igxObjectRange.ShadowFormat.Depth = 5
igxObjectRange.ShadowFormat.Color = vbBlue
```

See Also ShadowFormat object

iGrafx API Object Hierarchy

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

SnapToGrid Method

Syntax *ObjectRange*.**SnapToGrid**

Description The SnapToGrid method shifts the position and spacing of diagram objects in the ObjectRange by aligning them to the snap grid.

Example The following example creates an ObjectRange containing several shapes. The shapes are slightly misaligned. It then aligns the position and spacing of the shapes using the SnapToGrid method.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(2050, 2025)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(4000, 2000)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(6025, 2050)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Set the FillColor to blue for the shapes
MsgBox "Click OK to snap the shapes to the snap grid."
' Snap the shapes to the snap grid
igxObjectRange.SnapToGrid
MsgBox "Click OK to continue"
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

SourceArrowFormat Property

Syntax *ObjectRange*.**SourceArrowFormat**

Data Type ArrowFormat object (read-only, See Object Properties)

Description The SourceArrowFormat property returns an ArrowFormat object for the specified ObjectRange object.

Description The SourceArrowFormat property returns an ArrowFormat object for the specified ObjectRange object. Use the SourceArrowFormat object to copy source arrow styles from one object to another, or to access the source arrow format properties within an ObjectRange, such as Color, Size, and Style.

The SourceArrowFormat object returns values for color, size, and style properties, but if the ObjectRange contains more than one source arrow, only matching properties will have meaningful return values. Non-matching destination arrow properties return a value of -1.

Example The following example creates an ObjectRange containing three shapes and two connector lines. The connector line source arrows are different colors. It then sets all the arrows to the same color using the SourceArrowFormat property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnLine1 As ConnectorLine
Dim igxConnLine2 As ConnectorLine
Dim igxObjectRange1 As ObjectRange
Dim igxArrowFormat1 As ArrowFormat
' Create shapes on the active diagram.
Set igxShape1 = Application.ActiveDiagram.DiagramObjects. _
   AddShape(1440, 1440)
Set igxShape2 = Application.ActiveDiagram.DiagramObjects. _
   AddShape(1440 * 4, 1440)
Set igxShape3 = Application.ActiveDiagram.DiagramObjects. _
   AddShape(1440 * 6, 1440)
' Draw a connector line between shapes 1 and 2
Set igxConnLine1 = Application.ActiveDiagram.DiagramObjects. _
   AddConnectorLine(RouteType:=ixRouteRightAngle, _
   RouteFlag:=ixRouteFlagFindEdge, SourceShape:=igxShape1, _
   SourceDir:=ixDirEast, SourceConnectType:=ixConnectRelativeToShape, _
   DestShape:=igxShape2, DestDir:=ixDirWest, _
   DestConnectType:=ixConnectRelativeToShape)
' Get the ConnectorFormat object
With igxConnLine1.ConnectorFormat
   .SourceArrowFormat.Style = ixArrow3
   .SourceArrowFormat.Color = vbBlue
   .SourceArrowFormat.Size = 3
   .LineFormat.Color = vbRed
   .LineFormat.Width = 3
End With
' Draw a connector line between shapes 2 and 3
Set igxConnLine2 = Application.ActiveDiagram.DiagramObjects. _
   AddConnectorLine(RouteType:=ixRouteRightAngle, _
   RouteFlag:=ixRouteFlagFindEdge, SourceShape:=igxShape2, _
   SourceDir:=ixDirEast, SourceConnectType:=ixConnectRelativeToShape, _
```

```
   DestShape:=igxShape3, DestDir:=ixDirWest, _
   DestConnectType:=ixConnectRelativeToShape)
' Get the ConnectorFormat object
With igxConnLine2.ConnectorFormat
   .SourceArrowFormat.Style = ixArrow3
   .SourceArrowFormat.Color = vbRed
   .SourceArrowFormat.Size = 3
   .LineFormat.Color = vbRed
   .LineFormat.Width = 3
End With
' Create the ObjectRange objects
Set igxObjectRange1 = ActiveDiagram.MakeObjectRange
' Add the shapes and connector lines to the range
Call igxObjectRange1.Add(igxShape1.DiagramObject)
Call igxObjectRange1.Add(igxShape2.DiagramObject)
Call igxObjectRange1.Add(igxConnLine1.DiagramObject)
Call igxObjectRange1.Add(igxConnLine2.DiagramObject)
' Set the first ObjectRange's arrow format
Set igxArrowFormat1 = igxObjectRange1.SourceArrowFormat
' Use SourceArrowFormat to make all the arrows black
MsgBox "Click OK to make all the arrows black"
igxObjectRange1.SourceArrowFormat.Color = vbBlack
MsgBox "Click OK to continue."
```

See Also ArrowFormat object

iGrafx API Object Hierarchy

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

SpaceEvenly Method

Syntax *ObjectRange*.**SpaceEvenly**(*newVal* As IxSpace)

Description The SpaceEvenly method spaces all objects in the ObjectRange evenly apart from each other.

The *newVal* argument specifies (which direction to space the objects, and whether to space the object according to their edges or centers.) the method to use for spacing the objects in the object range. The IxSpace constant defines the valid values, and are listed in the following table.

Value	Name of Constant
0	ixAcrossCenters
1	ixDownCenters
2	ixAcrossEdges
3	ixDownEdges

Example The following example creates three shapes and selects two of the shapes to be assigned to an ObjectRange object. The SpaceEvenly method is then used to space the shapes in the object range evenly apart from each other.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Set the igxObjectRange variable to the ObjectRange object.
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape2.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Space the shapes in the ObjectRange evenly
MsgBox ("Click OK to invoke the SpaceEvenly method")
igxObjectRange.SpaceEvenly (ixAcrossEdges)
igxObjectRange.SpaceEvenly (ixDownEdges)
MsgBox ("Shapes moved. Click OK to continue")
```

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

ThreeDFormat Property

Syntax *ObjectRange*.**ThreeDFormat**

Data Type ThreeDFormat object (Read-Only, See Object Properties)

Description The ThreeDFormat property returns a ThreeDFormat object. This object is used to define the 3D formatting characteristics for an object range. The formatting hierarchy for shapes allows the developer to customize and manipulate shapes individually.

Example The following example creates three shapes and selects two of the shapes to be assigned to an ObjectRange object. The ObjectRange object's ThreeDFormat property is then used to set a 3D type and a depth.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxObjectRange As ObjectRange
' Create 3 shapes and assign the shape variables to the 3 Shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 2)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
' Set the igxObjectRange variable to the ObjectRange object
Set igxObjectRange = ActiveDiagram.MakeObjectRange
' Add 2 of the Shapes to the ObjectRange object
igxObjectRange.Add igxShape1.DiagramObject
igxObjectRange.Add igxShape3.DiagramObject
' Use the ThreeDFormat Property to select a 3D format and depth
MsgBox "Click OK to change the 3D format."
igxObjectRange.ThreeDFormat.Type = ixThreeD10
igxObjectRange.ThreeDFormat.Depth = 3
MsgBox "Click OK to continue"
```

See Also ThreeDFormat object

iGrafx API Object Hierarchy

{button ObjectRange object,JI(`igrafxrf.HLP',`ObjectRange_Object')}

OleObject Object

The OleObject object represents any object that is OLE compliant. OLE is Object Linking or Embedding. Word documents and Excel spreadsheets are examples of documents that can be inserted into a Diagram as an OleObject.

Use a *linked* object or an *embedded* object to insert a document created in another application. This becomes an OleObject in the Diagram. OleObjects are inserted into a Diagram using the **Insert->Ole Object…** menu item in iGrafx Professional.

If you create a new OleObject, it is *embedded*. If you add an OleObject from a file, you have the choice to make it *embedded*, or *linked*. The **Insert->Ole Object…** dialog box provides these options.The main differences between linked objects and embedded objects are where the data is stored, and how it is updated after you place it in the Diagram.

With a *linked* object, information is updated only if you modify the source file. Linked data is stored in the source file. The destination file stores only the location of the source file and displays a representation of the linked data. Use linked objects if file size is a consideration.

With an *embedded* object, information in the destination file does not change if you modify the source file. Embedded objects become part of the destination file and, once inserted, are no longer part of the source file. double-click the embedded object to open it in the source program.

Properties, Methods, and Events

All of the properties, methods, and events for the OleObject object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Activate	
ClassName	DoVerb	
DiagramObject	Edit	
Interface	Open	
Parent		
ProgID		

Activate Method

Syntax *OleObject*.**Activate** (*IOLEVerb* As Long)

Description The Activate method activates an OleObject, opening it for editing within the diagram.

The *IOLEVerb* argument specifies an OLE verb to execute. This argument gives the Activate method essentially the same functionality as the DoVerb method. See the DoVerb method for some common OLE verbs.

Example The following example activates an embedded Word document. To try this example you must first embed a new Word document in a diagram. Go to the iGrafx Professional interface and follow these steps:

1. Select the Insert->Ole Object… menu item
2. Check the Create New checkbox
3. Choose "Microsoft Word Document"
4. Click OK.

Next, run the following code to find the OLE object and activate it.

```
' Dimension the variables
Dim igxOle As OleObject
Dim igxObject As Object
' Find the Ole object
With ActiveDiagram.DiagramObjects
   ' Iterate through all the diagram objects
   For Index = 1 To .Count
      If .Item(Index).Type = ixObjectOle Then
         ' Once found, set the variable
         Set igxOle = .Item(Index).OleObject
         Exit For
      End If
   Next Index
End With
' Activate the Ole object
MsgBox "Click OK to activate the Ole Word document."
igxOle.Activate (-1)
MsgBox "Click OK to continue"
```

See Also DoVerb method

Edit method

Open method

{button OleObject object,JI(`igrafxrf.HLP',`OleObject_Object')}

ClassName Property

Syntax *OleObject*.**ClassName**

Data Type String (read-only)

Description The ClassName property returns a string containing the class name of the OleObject. For instance, if the OleObject is a Word document, the ClassName property returns the string "Microsoft Word Document". This property provides a way to identify the “type” of an object.

Example The following example finds the first OleObject in the diagram, and displays its class name. To try this example you must first embed a new Word document (or another Ole object) in the diagram. Go to the iGrafx Professional interface and follow these steps:

1. Select the Insert->Ole Object… menu item
2. Check the Create New checkbox
3. Choose "Microsoft Word Document"
4. Click OK.

Next, run the following code. The class name for your OLE object is retrieved and displayed.

```
' Dimension the variables
Dim igxOle As OleObject
Dim igxObject As Object
' Find the Ole object
With ActiveDiagram.DiagramObjects
   ' Iterate through all the diagram objects
   For Index = 1 To .Count
      If .Item(Index).Type = ixObjectOle Then
         ' Once found, set the variable
         Set igxOle = .Item(Index).OleObject
         Exit For
      End If
   Next Index
End With
' Display the class name of the Ole object
MsgBox "The class name is: " & igxOle.ClassName
```

See Also ProgID property

{button OleObject object,JI(`igrafxrf.HLP',`OleObject_Object')}

DiagramObject Property

Syntax *OleObject*.**DiagramObject**

Data Type DiagramObject object (read-only, See Object Properties)

Description The DiagramObject property returns the DiagramObject level (the “Extender” object) for the specified OleObject object. At this level, the developer has access to the DiagramObject attributes of the OleObject, such as size and position properties.

Example The following example finds the first OleObject in the diagram, and changes it's position and size. To try this example you must first embed a new Word document (or another Ole object) in the diagram. Go to the iGrafx Professional interface and follow these steps:

1. Select the Insert->Ole Object… menu item
2. Check the Create New checkbox
3. Choose "Microsoft Word Document"
4. Click OK.

Next, run the following code. The class name for your OLE object is retrieved and displayed, and the OleObject is resized and repositioned within the diagram.

```
' Dimension the variables
Dim igxOle As OleObject
Dim igxObject As Object
' Find the Ole object
With ActiveDiagram.DiagramObjects
   ' Iterate through all the diagram objects
   For Index = 1 To .Count
      If .Item(Index).Type = ixObjectOle Then
         ' Once found, set the variable
         Set igxOle = .Item(Index).OleObject
         Exit For
      End If
   Next Index
End With
' Display the class name of the Ole object
MsgBox "Click OK to reposition the OleObject."
' Reposition and resize the OleObject
With igxOle.DiagramObject
    .Width = 4000
    .Height = 3000
    .Top = 2000
    .Left = 2000
End With
' Put the OleObject in edit mode to view it's borders
igxOle.Edit
' Refresh the diagram
ActiveDiagram.Refresh
MsgBox "Click OK to continue"
```

See Also DiagramObject object

iGrafx API Object Hierarchy

{button OleObject object,JI(`igrafxrf.HLP',`OleObject_Object')}

DoVerb Method

Syntax *OleObject*.**DoVerb** (*IOLEVerb* As Long)

Description The DoVerb method specifies an OLE Verb to execute for the specified OleObject. The *IOLEVerb* argument is the Verb number to execute.

OLE objects recognize verb commands. All OLE objects have some verbs in common, which are expressed as negative numbers. OLE Verbs that are specific to an object are expressed as positive numbers. The following table lists some common OLE verbs.

Ole Verb	Result
0	Default verb, usually equivalent to Activate (-1)
-1	Activates the object for in place editing (no toolbars)
-2	Opens the object's parent application for editing the OleObject's contents
-3	For embedded objects, hides the application that created the object.
-4	Activates the OleObject for in-place editing with toolbars, if supported
-5	Activates the object for in-place editing when clicked with the mouse
-6	Clears the Undo history of the object, preventing it's application from Undoing your changes

Example The following example finds the first OleObject in the diagram, and changes it's position and size. To try this example you must first embed a new Word document (or another Ole object) in the diagram. Go to the iGrafx Professional interface and follow these steps:

1. Select the Insert->Ole Object… menu item
2. Check the Create New checkbox
3. Choose "Microsoft Word Document"
4. Click OK.

Next, run the following code. The class name for your OLE object is retrieved and displayed, and the OleObject is resized and repositioned within the diagram.

```
' Dimension the variables
Dim igxOle As OleObject
Dim igxObject As Object
' Find the Ole object
With ActiveDiagram.DiagramObjects
   ' Iterate through all the diagram objects
   For Index = 1 To .Count
      If .Item(Index).Type = ixObjectOle Then
         ' Once found, set the variable
         Set igxOle = .Item(Index).OleObject
         Exit For
      End If
   Next Index
End With
' Display the class name of the Ole object
MsgBox "Click OK to reposition the OleObject."
' Reposition and resize the OleObject
With igxOle.DiagramObject
```

```
    .Width = 4000
    .Height = 3000
    .Top = 2000
    .Left = 2000
End With
' Put the OleObject in edit mode to view it's borders
igxOle.Edit
' Refresh the diagram
ActiveDiagram.Refresh
MsgBox "Click OK to continue"
```

{button OleObject object,JI(`igrafxrf.HLP',`OleObject_Object')}

Edit Method

Syntax *OleObject*.**Edit**

Description The Edit method places an embedded OLE object in Edit mode for in-place editing. This allows the user to edit the contents of the OleObject without leaving the iGrafx Professional diagram. The Edit method is equivalent to DoVerb(-1). For this method to work, the OLE object must support in-place editing.

Example The following example finds the first OleObject in the diagram, and puts it into edit mode, using the Edit method. To try this example you must first embed a new Word document (or another Ole object) in the diagram. Go to the iGrafx Professional interface and follow these steps:

1. Select the Insert->Ole Object… menu item
2. Check the Create New checkbox
3. Choose "Microsoft Word Document"
4. Click OK.

Next, run the following code. The class name for your OLE object is retrieved and displayed, and the OleObject is resized and repositioned within the diagram.

```
' Dimension the variables
Dim igxOle As OleObject
Dim igxObject As Object
' Find the Ole object
With ActiveDiagram.DiagramObjects
   ' Iterate through all the diagram objects
   For Index = 1 To .Count
      If .Item(Index).Type = ixObjectOle Then
         ' Once found, set the variable
         Set igxOle = .Item(Index).OleObject
         Exit For
      End If
   Next Index
End With
' Hide the OleObject's edit window
igxOle.DoVerb (-3)
MsgBox "Click OK to edit the OleObject in-place."
' Put the OleObject in edit mode to view it's borders
igxOle.Edit
MsgBox "Click OK to continue"
```

{button OleObject object,JI(`igrafxrf.HLP',`OleObject_Object')}

Interface Property

Syntax *OleObject*.**Interface** As Object

Data Type Object (read-only)

Description The Interface property returns the OleObject's interface. This is relevant for OLE objects that are Visual Basic controls, such as command buttons, check boxes, Microsoft Grid objects, and other controls that supply an interface.

The Interface property provides access to the member functions of the OleObject. Properties and methods are different for various VB controls. Consult the documentation provided for the OLE object for more information.

Example The following example gets the interface for a Toggle Button control and changes the caption property, and the value of the control. This example requires at lease one Toggle Button inserted into the diagram. This is done from the Visual Basic Controls menu in iGrafx Professional.

```
' To run this example, first add a ToggleButton to the diagram.
' This is done from the Controls menu.
'
' Dimension the variables
Dim igxOle As OleObject
Dim igxToggle As ToggleButton
' Find the Ole object
With ActiveDiagram.DiagramObjects
    ' Iterate throught all the diagram objects
   For Index = 1 To .Count
       If .Item(Index).Type = ixObjectOle Then
           ' Once found, set the variable
           Set igxOle = .Item(Index).OleObject
           Exit For
       End If
   Next Index
End With
' Get the ToggleButton's interface
Set igxToggle = igxOle.Interface
' Turn off the toggle
igxToggle.Value = False
' Set the toggle's caption
MsgBox "Click OK to set the caption."
igxToggle.Caption = "Go"
' Turn on the toggle
MsgBox "Click OK to toggle the ToggleButton on."
igxToggle.Value = True
MsgBox "Click OK to continue"
```

{button OleObject object,JI(`igrafxrf.HLP',`OleObject_Object')}

Open Method

Syntax *OleObject*.**Open**

Description The Open method launches the specified OleObject object's parent software application, and opens the document. For instance, if the OleObject is an Excel spreadsheet, the Open method launches Microsoft Excel, and opens the document for editing. The Open method is equivalent to DoVerb(-2).

Example The following example finds the first OleObject in the diagram, and checks the class name. If it's a Word document, it opens the document in Word. To try this example you must first embed a new Word document (or another Ole object) in the diagram. Go to the iGrafx Professional interface and follow these steps:

1. Select the Insert->Ole Object… menu item
2. Check the Create New checkbox
3. Choose "Microsoft Word Document"
4. Click OK.

Next, run the following code. The class name for your OLE object is a Microsoft Word document, the the document is opened and can be edited. If it is not a Word document, then a message is displayed to indicate the OLE object is being skipped.

```
' Dimension the variables
Dim igxOle As OleObject
Dim igxObject As Object
' Find the Ole object
With ActiveDiagram.DiagramObjects
   ' Iterate through all the diagram objects
   For Index = 1 To .Count
      If .Item(Index).Type = ixObjectOle Then
         ' Once found, set the variable
         Set igxOle = .Item(Index).OleObject
         Exit For
      End If
   Next Index
End With
If igxOle.ClassName = "Microsoft Word Document" Then
    MsgBox "Click OK to launch Word for editing."
    igxOle.Open
Else
    MsgBox "Not a Word document. We'll skip it."
End If
MsgBox "Click OK to continue"
```

{button OleObject object,JI(`igrafxrf.HLP',`OleObject_Object')}

ProgID Property

Syntax *OleObject*.**ProgID**

Data Type String (read-only)

Description The ProgID property returns a string containing the application object ID. The string is in the form *appname.objecttype.* For instance, if the OleObject is a Word document, the ProgID property returns the string "word.document.8".

The string that ProdID returns is suitable for use as the *class* argument with Visual Basic's CreateObject function, for example:

```
Set MyVariable = CreateObject(OleObject.ProgID)
```

For more information, refer to the Visual Basic "CreateObject" and "GetObject" help topics.

Example The following example finds the first OleObject in the diagram, and displays it's ProgID. To try this example you must first embed a new Word document (or another Ole object) in the diagram. Go to the iGrafx Professional interface and follow these steps:

1. Select the Insert->Ole Object… menu item
2. Check the Create New checkbox
3. Choose "Microsoft Word Document"
4. Click OK.

Next, run the following code. The ProgID property for your OLE object is displayed.

```
' Dimension the variables
Dim igxOle As OleObject
Dim igxObject As Object
' Find the Ole object
With ActiveDiagram.DiagramObjects
   ' Iterate through all the diagram objects
   For Index = 1 To .Count
      If .Item(Index).Type = ixObjectOle Then
         ' Once found, set the variable
         Set igxOle = .Item(Index).OleObject
         Exit For
      End If
   Next Index
End With
MsgBox "The ProgID is: " & igxOle.ProgID
```

{button OleObject object,JI(`igrafxrf.HLP',`OleObject_Object')}

Path Object

The Path object is one segment of a flow path through a diagram or process. A Path defines where transactions and entities travel from object to object. When a connector line is created, a Path is created. Path objects provide properties for accessing associated ConnectorLine objects, CustomDataValue objects, Source and Destination objects (typically shapes), and the DecisionCase index.

Properties, Methods, and Events

All of the properties, methods, and events for the Path object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		
ConnectorLines		
CustomDataValues		
DecisionCaseIndex		
Destination		
Name		
Parent		
Source		

ConnectorLines Property

Syntax *Path*.**ConnectorLines**

Data Type ObjectRange object (read-only, See Object Properties)

Description The ConnectorLines property returns an ObjectRange object containing all the ConnectorLine objects involved in the path. Usually a path has only one ConnectorLine, but it is possible to string together ConnectorLines from the iGrafx Professional user interface (cannot be done through the APIs), in which case the ObjectRange contains more than one ConnectorLine object.

Exampe The following example creates three Shapes and two ConnectorLines in the diagram. It then uses the ConnectorLines property to get an ObjectRange containing all the ConnectorLines involved with Path 1 of Shape 1. To show the result, the color of the ConnectorLine is changed.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim Object As DiagramObject
Dim igxPath1 As Path
Dim igxPath2 As Path
Dim igxRange As ObjectRange
' Add several objects to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
igxShape1.Text = "Shape 1"
igxShape2.Text = "Shape 2"
igxShape3.Text = "Shape 3"
' Add connector lines
Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape2, _
    ixDirEast)
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, , igxShape1, ixDirNorth, , , , igxShape3, _
    ixDirWest)
' Get the path object from Shape 1
Set igxPath1 = igxShape1.OutputPaths.Item(1)
' Create a new object range
Set igxRange = ActiveDiagram.MakeObjectRange
' Get the Path's ConnectorLines as an ObjectRange
igxRange.AddRange igxPath1.ConnectorLines
' Bring the ObjectRange to the front
igxRange.Order ixBringToFront
' Change the color of the connectors for the Path
MsgBox "Change ConnectorLine color associated with Path 1 of Shape 1."
For Each Object In igxRange
   Object.ConnectorLine.LineColor = vbGreen
Next Object
MsgBox "Click OK to continue"
```

See Also

ObjectRange object

ConnectorLine object

iGrafx API Object Hierarchy

{button Path object,JI(`igrafxrf.HLP',`Path_Object')}

CustomDataValues Property

Syntax *Path*.**CustomDataValues**

Data Type CustomDataValues collection object (read-only, See Object Properties)

Description The CustomDataValues property returns the CustomDataValues collection for the specified Path object. The CustomDataValues collection provides access to all the CustomDataValue objects of the Path.

Example The following example gives the Path a time duration, which is reported to a CustomDataDefinition object. The accumulated time data is displayed in a Legend. The Path's duration value is displayed at the end of the procedure.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnector As ConnectorLine
Dim igxPath As Path
Dim igxDataDef As CustomDataDefinition
Dim igxValue1 As CustomDataValue
' Add shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
Set igxConnector = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape2, ixDirWest)
' Get the Path object from Shape 1
Set igxPath = igxShape1.OutputPaths.Item(1)
' Add a legend
ActiveDiagram.DiagramObjects.AddLegend 3000, 4000
' Add a CustomDataDefinition to the document
ThisDocument.CustomDataDefinitions.Add "MyTime", ixCustomDataFormatTimeBase
Set igxDataDef = ThisDocument.CustomDataDefinitions.Item(1)
' Get the shape's CustomDataValue object
Set igxValue1 = igxPath.CustomDataValues.Item(1, ixCustomDataDuration)
' Set the format for the time data
igxDataDef.Format = ixFieldFormatHMS
' Set the value of the Path's CustomDataValue
igxPath.CustomDataValues.Item(1, ixCustomDataDuration).Value _
    = "12:05:34"
' Report the time components of value
MsgBox "Path 1 duration: " & igxValue1.FormattedValue
```

See Also CustomDataValues object

iGrafx API Object Hierarchy

{button Path object,JI(`igrafxrf.HLP',`Path_Object')}

DecisionCaseIndex Property

Syntax *Path*.**DecisionCaseIndex**

Data Type Long (read-only)

Description The DecisionCaseIndex property returns a DecisionCaseIndex number. If the Path's source shape has two or more DecisionCases, the source Shape has a DecisioinCases collection object. Each DecisionCase in the collection has an index number. Each DecisionCase also has a path associated with it. The Path object's DecisionCaseIndex property returns the index number of it's associated DecisionCase.

Example The following example creates a shape with two DecisionCases. The Path object associated with each DecisionCase is set, and the DecisionCase name of each Path is displayed.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxConnector1 As ConnectorLine
Dim igxConnector2 As ConnectorLine
Dim Object As DiagramObject
Dim igxPath1 As Path
Dim igxPath2 As Path
Dim igxCase1 As DecisionCase
Dim igxCase2 As DecisionCase
' Add objects to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440 * 3, 1440 * 3)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 5, 1440)
igxShape1.Text = "Shape 1"
igxShape2.Text = "Shape 2"
igxShape3.Text = "Shape 3"
' Add connector lines
Set igxConnector1 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirWest, , , , igxShape2, _
    ixDirEast)
Set igxConnector2 = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteDirect, , igxShape1, ixDirEast, , , , igxShape3, _
    ixDirWest)
Set igxCase1 = igxShape1.DecisionCases.Add("Left")
Set igxCase2 = igxShape1.DecisionCases.Add("Right")
' Get Path 1 from Shape 1
Set igxPath1 = igxShape1.OutputPaths.Item(1)
' Get Path 2 from Shape 1
Set igxPath2 = igxShape1.OutputPaths.Item(2)
MsgBox "Path 1 Decision Case: " & _
    igxShape1.DecisionCases.Item(igxPath1.DecisionCaseIndex).Name _
    & Chr(13) & "Path 2 Decision Case: " & _
    igxShape1.DecisionCases.Item(igxPath2.DecisionCaseIndex).Name
```

See Also DecisionCase object

{button Path object,JI(`igrafxrf.HLP',`Path_Object')}

Destination Property

Syntax *Path*.**Destination**

Data Type Shape object (read-only, See Object Properties)

Description The Destination property returns the Shape object that is the destination of the specified Path object.

Example The following example creates four shapes in the active diagram, and connects them with connector lines. The diagram objects collection is then searched to find the Shape objects. When a Shape object is found, its OutputPaths collection is accessed and for each output path from the shape, the name of the Destination shape is listed in a message box.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxDiagramObjects As DiagramObjects
Dim igxConnLine As ConnectorLine
Dim igxPaths As Paths
' Create 4 shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape1.Text = "Shape One"
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 4)
igxShape2.Text = "Shape Two"
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440 * 4)
igxShape3.Text = "Shape Three"
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2.5, 1440 * 5)
igxShape4.Text = "Shape Four"
' Connect Shape1 to Shape2
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
    ixDirNorth, ixConnectRelativeToShape)
' Connect Shape1 to Shape3
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape3, _
    ixDirNorth, ixConnectRelativeToShape)
' Connect Shape2 to Shape3
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape3, _
    ixDirWest, ixConnectRelativeToShape)
' Connect Shape2 to Shape4
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
    ixDirWest, ixConnectRelativeToShape)
' Connect Shape3 to Shape4
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape3, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
    ixDirEast, ixConnectRelativeToShape)
```

```
' Connect Shape4 to Shape1
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape4, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape1, _
    ixDirEast, ixConnectRelativeToShape)
MsgBox "View the diagram"
' Get the Diagram objects collection of the Active Diagram
Set igxDiagramObjects = ActiveDiagram.DiagramObjects
' Find the Diagram Objects that are shapes
' List the text of the Destination shape of each output path for
' all the shapes in the diagram
For iCount = 1 To igxDiagramObjects.Count
    If (igxDiagramObjects.Item(iCount).Type = ixObjectShape) Then
        Set igxPaths = igxDiagramObjects.Item(iCount).Shape.OutputPaths
        For Index = 1 To igxPaths.Count
            MsgBox "Destination shape of output path named " _
                & igxPaths.Item(Index).Name & " is: " _
                & igxPaths.Item(Index).Destination.Text
        Next Index
    End If
Next iCount
```

See Also

Source property

Shape object

iGrafx API Object Hierarchy

{button Path object,JI(`igrafxrf.HLP',`Path_Object')}

Source Property

Syntax *Path*.**Source**

Data Type Shape object (read-only, See Object Properties)

Description The Source property returns the Shape object that is the source of the specified Path object.

Example The following example creates four shapes in the active diagram, and connects them with connector lines. The diagram objects collection is then searched to find the Shape objects. When a Shape object is found, its OutputPaths collection is accessed and for each output path from the shape, the name of the Source shape is listed in a message box.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxDiagramObjects As DiagramObjects
Dim igxConnLine As ConnectorLine
Dim igxPaths As Paths
' Create 4 shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape1.Text = "Shape One"
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 4)
igxShape2.Text = "Shape Two"
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440 * 4)
igxShape3.Text = "Shape Three"
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2.5, 1440 * 5)
igxShape4.Text = "Shape Four"
' Connect Shape1 to Shape2
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
    ixDirNorth, ixConnectRelativeToShape)
' Connect Shape1 to Shape3
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape3, _
    ixDirNorth, ixConnectRelativeToShape)
' Connect Shape2 to Shape3
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape3, _
    ixDirWest, ixConnectRelativeToShape)
' Connect Shape2 to Shape4
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
    ixDirWest, ixConnectRelativeToShape)
' Connect Shape3 to Shape4
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape3, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
    ixDirEast, ixConnectRelativeToShape)
' Connect Shape4 to Shape1
```

```
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape4, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape1, _
    ixDirEast, ixConnectRelativeToShape)
MsgBox "View the diagram"
' Get the Diagram objects collection of the Active Diagram
Set igxDiagramObjects = ActiveDiagram.DiagramObjects
' Find the Diagram Objects that are shapes
' List the text of the Source shape of each output path for
' all the shapes in the diagram
For iCount = 1 To igxDiagramObjects.Count
    If (igxDiagramObjects.Item(iCount).Type = ixObjectShape) Then
        Set igxPaths = igxDiagramObjects.Item(iCount).Shape.OutputPaths
        For Index = 1 To igxPaths.Count
            MsgBox "Source shape of output path named " _
                & igxPaths.Item(Index).Name & " is: " _
                & igxPaths.Item(Index).Source.Text
        Next Index
    End If
Next iCount
```

See Also

Destination property

Shape object

iGrafx API Object Hierarchy

{button Path object,JI(`igrafxrf.HLP',`Path_Object')}

Paths Object

The Paths object is a collection of Path objects. A Paths collection is associated with each Shape object. Its purpose is to store and provide access to the individual Path objects of shapes.

The Paths object provides the following functionality:

- The ability to access any Path object that has been created for a Shape object.
- The ability to determine how many Path objects are currently in the collection.

Properties, Methods, and Events

All of the properties, methods, and events for the Paths object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
Parent		

Related Topics

Path object

iGrafx API Object Hierarchy

Item Method

Syntax *Paths*.**Item**(*Index* As Integer) As Path

Description The Item method returns the Path object at the specified *Index* from the Paths collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Path. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. Use error trapping to handle these errors.

Example The following example creates four shapes in the active diagram, and connects them with connector lines. The diagram objects collection is then searched to find the Shape objects. When a Shape object is found, its OutputPaths collection is accessed and the names of all the output paths from the shape are listed in a message box using the Path.Name property.

```
' Dimension the variables
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxShape3 As Shape
Dim igxDiagramObjects As DiagramObjects
Dim igxConnLine As ConnectorLine
Dim igxPaths As Paths
' Create 4 shape objects
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape1.Text = "Shape One"
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 4)
igxShape2.Text = "Shape Two"
Set igxShape3 = ActiveDiagram.DiagramObjects.AddShape(1440 * 4, 1440 * 4)
igxShape3.Text = "Shape Three"
Set igxShape4 = ActiveDiagram.DiagramObjects.AddShape(1440 * 2.5, 1440 * 5)
igxShape4.Text = "Shape Four"
' Connect Shape1 to Shape2
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape2, _
    ixDirNorth, ixConnectRelativeToShape)
' Connect Shape1 to Shape3
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape3, _
    ixDirNorth, ixConnectRelativeToShape)
' Connect Shape2 to Shape3
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
    ixDirEast, ixConnectRelativeToShape, , , igxShape3, _
    ixDirWest, ixConnectRelativeToShape)
' Connect Shape2 to Shape4
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape2, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
    ixDirWest, ixConnectRelativeToShape)
' Connect Shape3 to Shape4
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape3, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape4, _
```

```
    ixDirEast, ixConnectRelativeToShape)
' Connect Shape4 to Shape1
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape4, _
    ixDirSouth, ixConnectRelativeToShape, , , igxShape1, _
    ixDirEast, ixConnectRelativeToShape)
MsgBox "View the diagram"
' Get the Diagram objects collection of the Active Diagram
Set igxDiagramObjects = ActiveDiagram.DiagramObjects
' Find the Diagram Objects that are shapes, and list their
' Output paths
For iCount = 1 To igxDiagramObjects.Count
    sOPaths = ""
    If (igxDiagramObjects.Item(iCount).Type = ixObjectShape) Then
        Set igxPaths = igxDiagramObjects.Item(iCount).Shape.OutputPaths
        For Index = 1 To igxPaths.Count
            sOPaths = sOPaths & igxPaths.Item(Index).Name & Chr(13)
        Next Index
        MsgBox igxDiagramObjects.Item(iCount).Shape.Text _
            & "'s Output paths are: " & Chr(13) & sOPaths
    End If
Next iCount
```

{button Paths object,JI(`igrafxrf.HLP',`Paths_Object')}

Property Object

The Property object is used for adding variables of any type you require to a Document object, Diagram object, or a DiagramObject object, such as a shape. A property can be used to store data that you want to collect, monitor, or use to drive other objects within a diagram or document.

Property objects are not, typically, user interface objects. Therefore, they provide a different level of functionality than do Field objects and “CustomData” objects.

Properties, Methods, and Events

All of the properties, methods, and events for the Property object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
Name		
Parent		
Value		

Related Topics

PropertyList object

PropertyLists object

iGrafx API Object Hierarchy

Value Property

Syntax *Property*.**Value**

Data Type Variant (read/write)

Description The Value property specifies the value (or the data) contained in the Property object. The Value property is of type Variant, which means it can contain string, date, time, boolean, or numeric values. For more information, see the Visual Basic documentation on the Variant data type.

Example The following example illustrates the use of the Value property.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPropertyList As PropertyList
Dim igxPropertyLists As PropertyLists
Dim igxProperty As Property
' Create a shape
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set the igxPropertyLists variable to the PropertyLists collection object
Set igxPropertyLists = igxShape.DiagramObject.PropertyLists
' Set the igxPropertyList variable to the PropertyList object
Set igxPropertyList = igxPropertyLists.Add("MyList")
' Create 3 Properties in the PropertyList
Set igxProperty = igxPropertyList.Add("propName")
Set igxProperty = igxPropertyList.Add("propNumber")
Set igxProperty = igxPropertyList.Add("propStatus")
' Set the values of the 3 Properties
igxPropertyList.Item(1).Value = "John Doe"
igxPropertyList.Item(2) = "1234"    ' .Value is optional
igxPropertyList.Item(3) = True
' Display the values of the 3 Properties
MsgBox ("propName is " & igxPropertyList.Item(1).Value)
MsgBox ("propNumber is " & igxPropertyList.Item(2))   ' .Value is implied
MsgBox ("propStatus is " & igxPropertyList.Item(3))
```

{button Property object,JI(`igrafxrf.HLP',`Property_Object')}

PropertyList Object

The PropertyList object is a collection of Property objects. A PropertyList collection is assessed through the PropertyLists collection of either a Document, a Diagram, or a DiagramObject. Any of these objects can have multiple PropertyList objects, which in turn can contain multiple Property objects. A PropertyList allows you to create properties that can be grouped according to any criteria to which you need to establish or adhere.

The PropertyList object provides the following functionality:

- The ability to access any Property objects that have been created for a Document, Diagram, or DiagramObject object.
- The ability to determine how many Property objects are currently in the collection.
- The ability to add and delete a Property object to or from the collection.
- A search capability for finding a specific property based on its name.
- The ability to query the name of the PropertyList object.

Properties, Methods, and Events

All of the properties, methods, and events for the PropertyList object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Delete	
Name	Item	
Parent	ItemExists	

Related Topics

Property object

PropertyLists object

iGrafx API Object Hierarchy

Add Method

Syntax *PropertyList*.**Add** (*PropertyName* As String) As Property

Description The Add method adds a new Property object to specified PropertyList object. The name of the new Property object is specified by the *PropertyName* argument. The method returns a Property object, so the result must be assigned to a variable of that type.

Errors IGRAFX_E_NAMEINUSE is returned if the supplied name is not unique within the property list.

Example The following example creates a shape, and then adds a PropertyList and eight properties to the shape's PropertyLists collection. It then displays a message giving the name of the property list, how many properties it contains, and the names of the properties.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPropertyLists As PropertyLists
Dim igxPropertyList As PropertyList
Dim igxProperty As Property
' Set igxShape variable to a new Shape object.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.Text = "My Shape"
' Set a igxPropertyLists variable for the shape
Set igxPropertyLists = igxShape.DiagramObject.PropertyLists
' Add a Property List and a property
Set igxPropertyList = igxPropertyLists.Add("Test List")
' Create 8 properties
For Index = 1 To 8
    Set igxProperty = igxPropertyList.Add("Test Property" & Index)
    sPropList = sPropList & igxProperty.Name & Chr(13)
Next Index
' Display the PropertyList name and the number of properties
MsgBox "The PropertyList, " & igxPropertyList.Name _
    & ", contains " & igxPropertyList.Count & " properties." _
    & Chr(13) & "These are:" & Chr(13) & sPropList
```

See Also ItemExists method

{button PropertyList object,JI(`igrafxrf.HLP',`PropertyList_Object')}

Item Method

Syntax *PropertyList*.**Item**(*Index* As Integer) As Property

Description The Item method returns the Property object at the specified *Index* from the PropertyList collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Property. An error is returned if the index is invalid.

Example The following example creates a shape, and then adds a PropertyList and a Property to the shape's PropertyLists collection. It then uses the Item method to access the first property and display it's name and the PropertyList name.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPropertyLists As PropertyLists
Dim igxPropertyList As PropertyList
Dim igxProperty As Property
' Set igxShape variable to a new Shape object.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.Text = "My Shape"
' Set a igxPropertyLists variable for the shape
Set igxPropertyLists = igxShape.DiagramObject.PropertyLists
' Add a Property List and a property
Set igxPropertyList = igxPropertyLists.Add("Test List")
igxPropertyList.Add "Test Property"
' Set the igxProperty variable to the Property just created
Set igxProperty = igxPropertyList.Item(1)
' Display the name of the PropertyList and the Property
MsgBox "PropertyList name is: " & igxPropertyList.Name _
    & Chr(13) & "Property name is: " & igxProperty.Name
```

See Also Add method

ItemExists method

{button PropertyList object,JI(`igrafxrf.HLP',`PropertyList_Object')}

ItemExists Method

Syntax *PropertyList*.**ItemExists** (*Name* As String) As Boolean

Description The ItemExists method searches the specified PropertyList collection for a property name. The name of the property is specified with the *Name* argument. The method returns a Boolean result indicating whether the property name is in the collection.

PropertyList collections can be associated with the Document, Diagram, and DiagramObject objects (refer to the discussion of the PropertyList object).

Example The following example creates a shape, and then adds a PropertyList and a Property to the shape's PropertyLists collection. It then asks the user to type a Property name to search for, using the ItemExists method.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPropertyLists As PropertyLists
Dim igxPropertyList As PropertyList
Dim igxProperty As Property
' Set igxShape variable to a new Shape object.
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.Text = "My Shape"
' Set a igxPropertyLists variable for the shape
Set igxPropertyLists = igxShape.DiagramObject.PropertyLists
' Add a Property List and a property
Set igxPropertyList = igxPropertyLists.Add("FiguresList")
Set igxProperty = igxPropertyList.Add("Cost")
' Check to see if the Property object is in the collection
For iLoop = 1 To 3
    If igxPropertyList.ItemExists(InputBox( _
        "Type a Property name to search for: ")) Then
        MsgBox "The Property object was found"
        Exit For
    Else
        MsgBox "That Property name is not in the collection."
    End If
 Next iLoop
```

{button PropertyList object,JI(`igrafxrf.HLP',`PropertyList_Object')}

PropertyLists Object

The PropertyLists object is a collection of PropertyList objects (which are also collections). A PropertyLists collection is associated with the following objects:

- Document
- Diagram
- DiagramObject

The collection stores and provides access to the individual PropertyList objects that have been created for any of the aforementioned objects. The PropertyLists object provides the following functionality:

- The ability to access any PropertyList objects that have been created for a Document, Diagram, or DiagramObject object.
- The ability to add a new PropertyList object to the collection.
- A search capability for finding a specific property list based on its name.

Important Notes

- The PropertyLists.Item method's Index argument only accepts a string value. Therefore, you must know the name of a property list in order to access it. This provides better security for PropertyList objects.
- You cannot use a For Each loop to iterate through the PropertyLists collection.

Properties, Methods, and Events

All of the properties, methods, and events for the PropertyLists object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Parent	Item	
	ItemExists	

Related Topics

Property object
PropertyList object
iGrafx API Object Hierarchy

Add Method

Syntax *PropertyLists*.**Add** (*PropertyListName* As String) As PropertyList

Description The Add method adds a PropertyList object to PropertyLists collection. The *PropertyListName* argument is used to provide a unique name for the PropertyList object within that collection. You can later use this name with the ItemExists method. If the name is not unique, then a run-time error occurs.

Example The following example creates a shape and gets its PropertyLists collection. The user is then prompted to enter a name for a PropertyList for the shape. The name is then displayed back to the user. A variation is also included for showing the result of entering a duplicate property list name.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPropertyListA As PropertyList
Dim igxPropertyListB As PropertyList
Dim igxPropertyListC As PropertyList
Dim igxPropertyLists As PropertyLists
Dim sReply As String
' Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.DiagramObject.ObjectName = "Example Shape"
igxShape.Text = "Test Shape"
' Set the igxPropertyLists variable to the PropertyLists
' collection associated with the shape (a DiagramObject)
Set igxPropertyLists = igxShape.DiagramObject.PropertyLists
' Allow the user to add a new PropertyList to the collection
sReply = InputBox("Enter a name for the new PropertyList:")
Set igxPropertyList = igxPropertyLists.Add(sReply)
MsgBox "A new PropertyList has been added to the shape." & _
    Chr(13) & "The name of the PropertyList is: " & _
    igxPropertyLists.Item(1).Name
```

Add the following lines at the end of the example. Then enter the same name, MyProps” in the Input Box.

```
' Add a second PropertyList object for the shape
Set igxPropertyList = igxPropertyLists.Add("MyProps")
```

See Also Item method

ItemExists method

{button PropertyLists object,JI(`igrafxrf.HLP',`PropertyLists_Object')}

Item Method

Syntax *PropertyLists*.**Item**(*Index* As String) As PropertyList

Description The Item method returns the PropertyList object at the specified *Index* from the PropertyLists collection. The data type of the *Index* argument is String, meaning that you must know the name of a property list in order to access it from the collection. The result of the method must be assigned to a variable of type PropertyList. An error is returned if the index is invalid.

Example The following example creates several new PropertyList objects for a shape, and then displays the name of each PropertyList using the Item method.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPropertyListA As PropertyList
Dim igxPropertyListB As PropertyList
Dim igxPropertyListC As PropertyList
Dim igxPropertyLists As PropertyLists
Dim sResults As String
' Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.DiagramObject.ObjectName = "Example Shape"
igxShape.Text = "Test Shape"
' Set the igxPropertyLists variable to the PropertyLists
' collection associated with the shape (a DiagramObject)
Set igxPropertyLists = igxShape.DiagramObject.PropertyLists
' Add 3 PropertyList objects to the collection
Set igxPropertyListA = igxPropertyLists.Add("Figures")
Set igxPropertyListB = igxPropertyLists.Add("Dates")
Set igxPropertyListC = igxPropertyLists.Add("Times")
' Get the names of the PropertyList objects in the shape's
' PropertyLists collection
For Index = 1 To igxPropertyLists.Count
    sResults = sResults & igxPropertyLists.Item(Index).Name _
    & Chr(13)
Next Index
MsgBox "The PropertyList objects defined for this DiagramObject are:" _
   & Chr(13) & Chr(13) & sResults
```

{button PropertyLists object,JI(`igrafxrf.HLP',`PropertyLists_Object')}

ItemExists Method

Syntax *PropertyLists*.**ItemExists** (*Name* As String) As Boolean

Description The ItemExists method searches the specified PropertyLists collection for a property list name. The name of the property list is specified with the *Name* argument. The method returns a Boolean result indicating whether the property list name is in the collection.

PropertyLists collections can be associated with the Document, Diagram, and DiagramObject objects (refer to the discussion of the PropertyLists object).

Description The following example adds some PropertyList objects to the PropertyLists collection associated with the shape DiagramObject. and then allows the user to type a property name to search using the ItemExists method.

```
' Dimension the variables
Dim igxShape As Shape
Dim igxPropertyListA As PropertyList
Dim igxPropertyListB As PropertyList
Dim igxPropertyListC As PropertyList
Dim igxPropertyLists As PropertyLists
Dim sResults As String
'Set igxShape variable to a new Shape object
Set igxShape = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
igxShape.DiagramObject.ObjectName = "Example Shape"
igxShape.Text = "Test Shape"
' Set the igxPropertyLists variable to the PropertyLists
' collection associated with the shape (a DiagramObject)
Set igxPropertyLists = igxShape.DiagramObject.PropertyLists
' Add 3 PropertyList objects to the collection
Set igxPropertyListA = igxPropertyLists.Add("Figures")
Set igxPropertyListB = igxPropertyLists.Add("Dates")
Set igxPropertyListC = igxPropertyLists.Add("Times")
' Allow the user to try 3 searches for a property list
For Index = 1 To 3
    ' An InputBox is used here to supply the Name argument
    ' for the ItemExists method. If the supplied name is found,
    ' the expression returns True
    If igxPropertyLists.ItemExists(InputBox _
        ("Enter a PropertyList name to search:")) Then
        MsgBox "There is a PropertyList with that name in the collection"
    Else
        MsgBox "No PropertyList in the collection with that name"
    End If
Next Index
```

{button PropertyLists object,JI(`igrafxrf.HLP',`PropertyLists_Object')}

Template Object

The Template object represents an iGrafx Professional document that has been stored specifically as a template. Templates can be used to set up particular types of diagrams, store special or custom shapes, etc. They serve the same purpose as a template in, for example, a word processing application. The Template object provides properties for identifying and locating a template, and a method for opening a template.

The Templates collection is accessed through the Application object. It allows you to set up a default path for template storage and access to Template objects.

Properties, Methods, and Events

All of the properties, methods, and events for the Template object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	OpenAsDocument	
FullName		
Name		
Parent		
Path		

Related Topics

Templates object

iGrafx API Object Hierarchy

OpenAsDocument Method

Syntax *Template*.**OpenAsDocument** As Document

Description The OpenAsDocument method opens a template as a document rather than invoking it as a template.

Example The following example attempts to find the Cascade.igt template. If the template is found, it is opened as a document.

```
' Dimension the variables
Dim Success As Boolean
' Set which folder to look for template files
Templates.DefaultTemplatePath = _
    "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\"
' Find the Cascade.igt template
For Index = 1 To Templates.Count
   If Templates.Item(Index).Name = "Cascade.igt" Then
      Templates.Item(Index).OpenAsDocument
      ' Flag that it was found
      Success = True
      Exit For
   End If
Next Index
' Report success or failure to find the template
If Success Then
   MsgBox "Cascade template opened as a document."
Else
   MsgBox "Cascade template not found."
End If
```

{button Template object,JI(`igrafxrf.HLP',`Template_Object')}

Templates Object

The Templates object is a collection of Template objects, and is accessible only from the Application object. Its purpose is to store and provide access to the individual Template objects.

The Templates object provides the following functionality:

- The ability to access any Template objects that have been created.
- The ability to determine how many Template objects are in the collection.
- The ability to set the default directory path for finding a specified template.

Properties, Methods, and Events

All of the properties, methods, and events for the Templates object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Item	
Count		
DefaultTemplatePath		
Parent		

Related Topics

Template object

iGrafx API Object Hierarchy

DefaultTemplatePath Property

Syntax *Templates*.**DefaultTemplatePath**

Data Type String (read/write)

Description The DefaultTemplatePath property specifies the file system folder where iGrafx Professional first looks for a Template file.

Example The following example sets the default path where iGrafx Professional first looks for template files. It then attempts to find the Cascade.igt template. If the template is found, it is opened as a document.

```
' Dimension the variables
Dim Success As Boolean
' Set which folder to look for template files
Templates.DefaultTemplatePath = _
   "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\"
' Find the Cascade.igt template
For Index = 1 To Templates.Count
   If Templates.Item(Index).Name = "Cascade.igt" Then
      Templates.Item(Index).OpenAsDocument
      ' Flag that it was found
      Success = True
      Exit For
   End If
Next Index
' Report success or failure to find the template
If Success Then
   MsgBox "Cascade template opened as a document."
Else
   MsgBox "Cascade template not found."
End If
```

Item Method

Syntax *Templates*.**Item**(*Index* As Integer) As Template

Description The Item method returns the Template object at the specified *Index* from the Templates collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type Template. An error is returned if the index is invalid.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is helpful to use error trapping before attempting to use the Item method.

Example The following example attempts to find the Cascade.igt template by iterating through the Templates collection object, using the Item method. If the template is found, it is opened as a document.

```
' Dimension the variables
Dim Success As Boolean
' Set which folder to look for template files
Templates.DefaultTemplatePath = _
   "C:\Program Files\iGrafx\Pro\8.0\Template\iGrid\"
' Find the Cascade.igt template
For Index = 1 To Templates.Count
   If Templates.Item(Index).Name = "Cascade.igt" Then
      Templates.Item(Index).OpenAsDocument
      ' Flag that it was found
      Success = True
      Exit For
   End If
Next Index
' Report success or failure to find the template
If Success Then
   MsgBox "Cascade template opened as a document."
Else
   MsgBox "Cascade template not found."
End If
```

{button Templates object,JI(`igrafxrf.HLP',`Templates_Object')}

ConnectPoint Object

The ConnectPoint object represents a connection point for a shape/shapeclass. A connect point is a defined location within the shape's relative coordinate space where connector lines or callout lines can be attached

A connect point is represented on a shape by small green circles, which are visible when a connector or callout line is being dragged over the relative coordinate space of a shape.

Properties, Methods, and Events

All of the properties, methods, and events for the ConnectPoint object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Delete	
Parent		
X		
Y		

X Property

Syntax *ConnectPoint*.**X**

Data Type Double (read/write)

Description The X property specifies the position of a connect point in the X direction. Use this property (along with the Y property) to either set or determine the position of a connect point on a shape. Valid values are typically 0.0 to 1.0, using the relative coordinate space of the shape. However, since the relative coordinate space can be redefined, the range of valid values can vary.

Example The following example demonstrates moving a connect point using the X and Y properties.

```
' Dimension the variables
Dim igxDiagramObject As DiagramObject
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxCnctPoint As ConnectPoint
' Add a shape to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440 + 360, 1440 * 3)
' Set a variable to the first connect point on the shape
Set igxCnctPoint = igxShape1.ShapeClass.ConnectPoints.Item(1)
' Move the point below the shape, down and to the right
igxCnctPoint.X = 1.5
igxCnctPoint.Y = 1.5
MsgBox "Connect point moved below and to right of first shape.  To " _
   & Chr(13) & "check it's position, return to the diagram, " _
   & "and drag a connector line " & Chr(13) & "from Shape1 to " _
   & "Shape2. As you drag near shape 1, the connect points appear " _
   & Chr(13) & "as green dots on and below the shape."
```

See Also Y property

Shape Coordinate Space

{button ConnectPoint object,JI(`igrafxrf.HLP',`ConnectPoint_Object')}

Y Property

Syntax *ConnectPoint*.**Y**

Data Type Double (read/write)

Description The Y property specifies the position of a connect point in the Y direction. Use this property (along with the X property) to either set or determine the position of a connect point on a shape. Valid values are typically 0.0 to 1.0, using the relative coordinate space of the shape. However, since the relative coordinate space can be redefined, the range of valid values can vary.

Example See the example for the X property.

See Also X property

Shape Coordinate Space

{button ConnectPoint object,JI(`igrafxrf.HLP',`ConnectPoint_Object')}

ConnectPoints Object

The ConnectPoints object is a collection of ConnectPoint objects that identify all the connection points for a ShapeClass object. Each ShapeClass object has its own separate ConnectPoints collection.

The ConnectPoints object provides the following functionality:

- The ability to access any ConnectPoint in the collection.
- The ability to determine how many ConnectPoint objects are in the collection.
- The ability to delete all of the ConnectPoints from the collection.
- The ability to add a new ConnectPoint to the collection.
- The ability to regenerate all of the default connection points for a shape or shapeclass.

Properties, Methods, and Events

All of the properties, methods, and events for the ConnectPoints object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	DeleteAll	
Parent	GenerateDefaultConnectPoints	
	Item	

Add Method

Syntax *ConnectPoints*.**Add** (*X* As Double, *Y* As Double) As ConnectPoint

Description The Add method adds a new connect point to a shape through its ShapeClass. The *X* and *Y* arguments allow you to specify the location for the new connect point. The arguments use current relative coordinate space for the shape. The default coordinate space for any shape is 0.0 to 1.0 along the X axis, and 0.0 to 1.0 along the Y axis; however, shape coordinate spaces can be modified. The method returns a ConnectPoint object, so the result of the method must be assigned to a variable of that type.

Example The following example adds a new connect point to the center of a shape, and thus, its shape class.

```
' Dimension the variables
Dim igxCnctPoint As ConnectPoint
Dim igxCnctPoints As ConnectPoints
Dim igxDiagramObject As DiagramObject
Dim igxShape1 As Shape
Dim igxShape2 As Shape
Dim igxConnLine As ConnectorLine
' Add two shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440 * 3)
Set igxCnctPoints = igxShape1.ShapeClass.ConnectPoints
' Put a new connect point in the middle of the shape
Set igxCnctPoint = igxShape1.ShapeClass.ConnectPoints.Add(0.5, 0.5)
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirEast, _
    ixConnectRelativeToShape)
MsgBox "To view the new ConnectPoint added to the shape, return to" _
    & Chr(13) & "the diagram, and drag the end of the connector line" _
    & " toward the middle of the shape.  In this" _
    & Chr(13) & "case, the new ConnectPoint appears as a blue" _
    & " dot to the right" & Chr(13) & "in the middle of the shape."
For Each ConnectPoint In igxCnctPoints
    Output Str(ConnectPoint.X) & ", " & Str(ConnectPoint.Y)
Next ConnectPoint
```

{button ConnectPoints object,JI(`igrafxrf.HLP',`ConnectPoints_Object')}

DeleteAll Method

Syntax *ConnectPoints*.**DeleteAll**

Description The DeleteAll method deletes all the connect points of the specified ShapeClass object. From a programmatic viewpoint, the method clears the ConnectPoints collection array of the ShapeClass object. Note that this method deletes ALL connect points, including the standard default connect points. The method affects all shapes in the diagram that belong to the specified ShapeClass object.

This method is useful if you want to reset the standard connect points for a shape class—use this method, and then use the GenerateDefaultConnectPoints method.

Example The following example deletes all the connection points on a shape.

```
' Dimension the variables
Dim igxCnctPoint As ConnectPoint
Dim igxDiagramObject As DiagramObject
Dim igxShape1 As Shape
Dim igxShape2 As Shape
' Add two shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 + 720, 1440 * 2)
' Delete all the connect points from both shapes
Call igxShape1.ShapeClass.ConnectPoints.DeleteAll
Call igxShape2.ShapeClass.ConnectPoints.DeleteAll
MsgBox "All the connection points on the shapes have been deleted."
' Attempt to connect the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirEast, _
    ixConnectRelativeToShape)
MsgBox "View the diagram"
MsgBox "All connect points in the shape have been deleted. To view the " _
    & Chr(13) & "results, return to the diagram, and drag a connector line to
or from " _
    & Chr(13) & "the shape.  In this case, no blue dot will appear on the
shape, " _
    & Chr(13) & "showing that there are no connect points."
' Restore the default connection points on the shape
Call igxShape2.ShapeClass.ConnectPoints.GenerateDefaultConnectPoints
MsgBox "The default connect points for the shape have been restored"
```

GenerateDefaultConnectPoints Method

Syntax *ConnectPoints*.**GenerateDefaultConnectPoints**

Description The GenerateDefaultConnectPoints method adds the default set of connect points to the ConnectPoints collection of a ShapeClass object. This method is useful for resetting the ConnectPoints collection after the ConnectPoints.DeleteAll method has been used.

Example The following example deletes all the connection points on a shape using the ConnectionPoints.DeleteAll method, and then restores the default set of connection points for the object using the GenerateDefaultConnectionPoints method.

```
' Dimension the variables
Dim igxCnctPoint As ConnectPoint
Dim igxConnLine As ConnectorLine
Dim igxDiagramObject As DiagramObject
Dim igxShape1 As Shape
Dim igxShape2 As Shape
' Add two shapes to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
Set igxShape2 = ActiveDiagram.DiagramObjects.AddShape _
    (1440 + 720, 1440 * 2)
' Delete all the connect points from Shape2
Call igxShape2.ShapeClass.ConnectPoints.DeleteAll
MsgBox "All the connection points on the shape have been deleted."
' Attempt to connect the two shapes
Set igxConnLine = ActiveDiagram.DiagramObjects.AddConnectorLine _
    (ixRouteRightAngle, ixRouteFlagFindEdge, igxShape1, ixDirEast, _
    ixConnectRelativeToShape, , , igxShape2, ixDirEast, _
    ixConnectRelativeToShape)
MsgBox "View the diagram"
' Restore the default connection points on the shape
Call igxShape2.ShapeClass.ConnectPoints.GenerateDefaultConnectPoints
MsgBox "The default connect points for the shape have been restored"
```

{button ConnectPoints object,JI(`igrafxrf.HLP',`ConnectPoints_Object')}

Item Method

Syntax *ConnectPoints*.**Item**(*Index* As Integer) As ConnectPoint

Description The Item method returns the ConnectPoint object at the specified *Index* from the ConnectPoints collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type ConnectPoint.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is helpful to use error trapping before attempting to use the Item method.

Example The following example uses the Item method to iterate through all the connect points in a shape, and display the position of each connect point, in coordinates.

```
' Dimension the variables
Dim igxCnctPoints As ConnectPoints
Dim igxDiagramObject As DiagramObject
Dim igxShape1 As Shape
Dim sResults As String
Dim Index As Integer
' Add a shape to the diagram
Set igxShape1 = ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
' Set a variable to the ConnectPoints object
Set igxCnctPoints = igxShape1.ShapeClass.ConnectPoints
' Iterate through all the connect points using the Item method
' and add each coordinate to a string
For Index = 1 To igxCnctPoints.Count
    sResults = sResults & Index & ":   " & _
        Round(igxCnctPoints.Item(Index).X, 2) & ",  " & _
        Round(igxCnctPoints.Item(Index).Y, 2) & Chr(13)
Next Index
MsgBox "Coordinates of all the connect points on the shape are: " _
    & Chr(13) & Chr(13) & sResults
```

{button ConnectPoints object,JI(`igrafxrf.HLP',`ConnectPoints_Object')}

Extension Object

An Extension object is part of an Extension Project. Extension Projects contain two project items: "ThisApplication", and "ThisExtension". The "ThisExtension" project item is associated with the Extension object.

The purpose of the Extension object is to listen to two events: Initialize, and Terminate. These events occur when an Extension project is loaded and unloaded.

Properties, Methods, and Events

All of the properties, methods, and events for the Extension object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application		Initialize
Parent		Terminate

Related Topics

ExtensionProject object

ExtensionProjects object

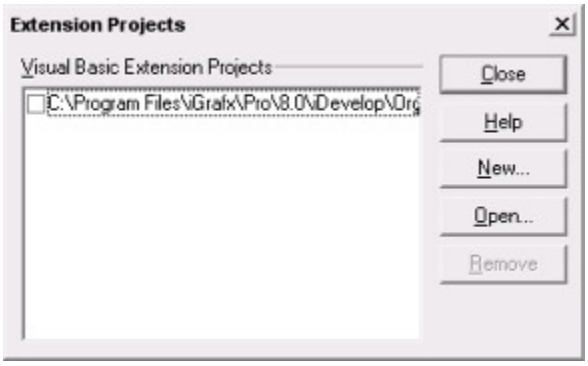
IGrafxExtension object

iGrafx API Object Hierarchy

Initialize Event

Syntax **Private Sub** *Extension*_**Initialize**()

Description The Initialize event occurs when an ExtensionProject file is loaded. Extension Projects occurs when the user uses the extension projects dialog to load an extension project (shown below), or the Extensions.Add or the ExtensionProject.Load methods. It also occurs on all installed extension projects when the application starts up.



Example The following example uses the Initialize and Terminate events to report when the Extension Project is Initialized and Terminated. This event code should go into the code pane called "ThisExtension".

```
Private Sub Extension_Initialize()
    MsgBox "The project has initialized." & Chr(13)
End Sub

Private Sub Extension_Terminate()
    MsgBox "The project has terminated."
End Sub
```

{button Extension object,JI(`igrafxrf.HLP',`Extension_Object')}

Terminate Event

Syntax **Private Sub** *Extension*_**Terminate**()

Description The Terminate event occurs when an ExtensionProject is unloaded or removed using the extension project dialog, or the ExtensionProject.Unload or ExtensionProject.Remove methods. The event also occurs on all installed extension projects when the application shuts down.

Example The following example uses the Initialize and Terminate events to report when the Extension Project is Initialized and Terminated. This event code should go into the code pane called "ThisExtension".

```
Private Sub Extension_Initialize()
    MsgBox "The project has initialized." & Chr(13)
End Sub

Private Sub Extension_Terminate()
    MsgBox "The project has terminated."
End Sub
```

{button Extension object,JI(`igrafxrf.HLP','Extension_Object')}

IGrafxExtension Object

The IGrafxExtension object is an interface that is implemented by a Class and registered as an extension. When a Class implements IGrafxExtension, it provides methods that determine what happens when the IGrafxExtension enters and leaves a particular iGrafx Professional context. iGrafx Professional contexts are defined in the iGrafx Extension Architecture. The extension architecture is beyond the scope of this help file, but will be described in detail in the *iGrafx Extensions Development Guide*.

Properties, Methods, and Events

All of the properties, methods, and events for the IGrafxExtension object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
	ContextBegin	
	ContextEnd	

Related Topics

Extension object

ExtensionProject object

ExtensionProjects object

iGrafx API Object Hierarchy

ContextBegin Method

Syntax *IGrafxExtension*.**ContextBegin**(*Host* As IxFlowExtensionHost)

Description The ContextBegin method appears in a Class that implements the IGrafxExtension interface. The ContextBegin method is designed by the programmer, and determines what happens when the IGrafxExtension enters the iGrafx Professional context that the extension has been registered for.

The Host argument is a “Flow Extension Host object”, which provides a gateway to the iGrafx Professional Extension architecture. The iGrafx System Developer’s Guide contains some information about the use of the iGrafxExtension object and its two methods.

Refer to the Application.RegisterExtension method for more information.

{button IGrafxExtension object,JI(`igrafxrf.HLP',`IGrafxExtension_Object')}

ContextEnd Method

Syntax *IGrafxExtension*.**ContextEnd**

Description The ContextEnd method appears in a Class that implements the IGrafxExtension interface. The ContextEnd method is designed by the programmer, and determines what happens when the IGrafxExtension leaves the iGrafx Professional context that the extension has been registered for.

Refer to the Application.RegisterExtension method for more information.

{button IGrafxExtension object,JI(`igrafxrf.HLP',`IGrafxExtension_Object')}

ExtensionProject Object

An ExtensionProject object is an application level code module, called an Extension Project. Extension Projects are useful for writing application level extensions to iGrafx Professional.

An Extension Project contains two project items: an Application project item, and an Extension project item. Programmers can add additional modules and classes to an Extension Project if desired.

The Application project item is named "ThisApplication". Using this project item, you can listen to application events, such as Startup, Quit, and BeforeWelcome. Also, the "ThisApplication" project item has AnyControls that let you listen to all other events, including any document events, any diagram events, any shape events, any connector events, etc.

The Extension project item is called "ThisExtension". This project item has two events: Initialize and Terminate. These events allow the programmer to determine what happens when an ExtensionProject is loaded and unloaded.

Extension Projects cannot be created or deleted using Visual Basic. Creating a new Extension Project must be performed from the Tools menu in the iGrafx Professional user interface:

Tools->Visual Basic->Extension Projects…->New

From Visual Basic, Extension Projects can be added to the collection, loaded, unloaded, and removed from the collection. Extension Projects that are loaded using the Load method are marked as "installed", and load every time you start the iGrafx Professional application. To prevent an Extension Project from loading automatically, use the Unload method.

Properties, Methods, and Events

All of the properties, methods, and events for the ExtensionProject object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
AnyControls	Load	
Application	Remove	
FullName	Unload	
Loaded		
Parent		
Path		
VisibleInIDE		

Related Topics

Extension object

ExtensionProjects object

IGrafxExtension object

iGrafx API Object Hierarchy

AnyControls Property

Syntax *ExtensionProject*.**AnyControls**

Data Type AnyControls object (read-only, See Object Properties)

Description The AnyControls property returns an AnyControls object. This allows the extension project to access the AnyControls that are associated with the Extension Project's Application project item.

A complete discussion of the AnyControls object is provided under that topic.

See Also AnyControls object

iGrafx API Object Hierarchy

Load Method

Syntax *ExtensionProject*.**Load**

Description The Load method makes an ExtensionProject active. The Load and Unload methods determine which ExtensionProject objects are active. Before using the Load method, an ExtensionProject resides on disk, but it is not visible in Visual Basic, and it is not running. After using the Load method, the Extension Project becomes visible, and is active for adding and running code. Also, an ExtensionProject that has been loaded will then load automatically every time iGrafx Professional starts.

Example The following example loads, and then unloads an Extension Project. This example requires at least one existing Extension Project. If necessary, create a new Extension Project from the iGrafx Professional Tools menu:

Tools->Visual Basic->Extension Projects…->New

```
' Dimension the variables
Dim igxProject As ExtensionProject
' Get the first Extension Project object
Set igxProject = ExtensionProjects.Item(1)
' Load the Extension Project
MsgBox "Click OK to load the Extension Project"
igxProject.Load
' Unload the Extension Project
MsgBox "Click OK to unload the Extension Project"
igxProject.Unload
```

See Also Loaded property

Unload method

{button ExtensionProject object,JI(`igrafxrf.HLP',`ExtensionProject_Object')}

Loaded Property

Syntax *ExtensionProject*.**Loaded**[= {**True | False**}]

Data Type Boolean (read-only)

Description The Loaded property indicates whether the specified ExtensionProject object is loaded. Refer to the Load method for more information.

Example The following example loads, and then unloads an Extension Project. It first checks the Loaded property, and loads the Project only if it is not yet loaded. This example requires at least one existing Extension Project. If necessary, create a new Extension Project from the iGrafx Professional Tools menu:

Tools->Visual Basic->Extension Projects…->New

```
' Dimension the variables
Dim igxProject As ExtensionProject
' Get the first Extension Project object
Set igxProject = ExtensionProjects.Item(1)
' If it's not yet loaded, load the Extension Project
If Not igxProject.Loaded Then
   MsgBox "Click OK to load the Extension Project"
   igxProject.Load
End If
' Unload the Extension Project
MsgBox "Click OK to unload the Extension Project"
igxProject.Unload
```

See Also Load method

Unload method

{button ExtensionProject object,JI(`igrafxrf.HLP',`ExtensionProject_Object')}

Remove Method

Syntax *ExtensionProject*.**Remove**

Description The Remove method removes an ExtensionProject from the ExtensionProjects collection. After using the Remove method, the Extension Project remains as a file on disk, but is not in the ExtensionProjects collection until you use the Load method to load it again. To unload an extension but still leave it in the ExtensionProjects collection, use the Unload method instead.

Example The following example asks the user for confirmation before removing all Extension Projects from the collection. This example requires at least one existing Extension Project. If necessary, create a new Extension Project from the iGrafx Professional Tools menu:

Tools->Visual Basic->Extension Projects…->New

```
' Dimension the variables
Dim igxProject As ExtensionProject
' Ask the user if they want to remove them all
If MsgBox("Remove all Extension Projects?", vbYesNo) = vbYes Then
   For Index = 1 To ExtensionProjects.Count
      ' Remove the Extension Project
      ExtensionProjects.Item(Index).Remove
   Next Index
End If
```

See Also ExtensionProjects.Add method

{button ExtensionProject object,JI(`igrafxrf.HLP',`ExtensionProject_Object')}

Unload Method

Syntax *ExtensionProject*.**Unload**

Description The Unload method makes the Extension Project inactive. After using the Unload method, the Extension Project remains as part of the current ExtensionProjects collection, but it is not active. It is no longer visible in Visual Basic, and is not accessible for running code. To unload and remove an extension project from the ExtensionProjects collection, use the Remove method.

Example The following example loads, and then unloads an Extension Project. This example requires at least one existing Extension Project. If necessary, create a new Extension Project from the iGrafx Professional Tools menu:

Tools->Visual Basic->Extension Projects…->New

```
' Dimension the variables
Dim igxProject As ExtensionProject
' Get the first Extension Project object
Set igxProject = ExtensionProjects.Item(1)
' Load the Extension Project
MsgBox "Click OK to load the Extension Project"
igxProject.Load
' Unload the Extension Project
MsgBox "Click OK to unload the Extension Project"
igxProject.Unload
```

See Also Load method

Loaded property

{button ExtensionProject object,JI(`igrafxrf.HLP',`ExtensionProject_Object')}

VisibleInIDE Property

Syntax *ExtensionProject*.**VisibleInIDE**[= {**True | False**}]

Data Type Boolean (read/write)

Description The VisibleInIDE property specifies whether the Extension Project is accessible in Visual Basic's Integrated Development Environment (IDE). If set to True, when an extension project is loaded, it will not be visible in the Visual Basic IDE.

If you develop an Extension Project and you don't want users to see your code (or have a lot of extension projects cluttering up the Visual Basic IDE), set the VisibleInIDE property to False to hide your extension project.

Extension Projects that are hidden in the Visual Basic IDE are listed in the extension projects dialog.

Example The following example has two subroutines. One make the ExtensionProject not visible, the other makes it visible. This example requires at least one existing Extension Project. If necessary, create a new Extension Project from the iGrafx Professional Tools menu:

Tools->Visual Basic->Extension Projects…->New

```
Private Sub MakeInvisible()
    ' Dimension the variables
    Dim igxProject As ExtensionProject
    ' Get the first Extension Project object
    Set igxProject = ExtensionProjects.Item(1)
    ' Load the Extension Project
    MsgBox "Click OK to make the Extension Project not visible"
    igxProject.VisibleInIDE = False
    igxProject.Unload
    igxProject.Load
End Sub

Private Sub MakeVisible()
    ' Dimension the variables
    Dim igxProject As ExtensionProject
    ' Get the first Extension Project object
    Set igxProject = ExtensionProjects.Item(1)
    ' Load the Extension Project
    ' Unload the Extension Project
    MsgBox "Click OK to make the Extension Project visible"
    igxProject.VisibleInIDE = True
    igxProject.Unload
    igxProject.Load
End Sub
```

{button ExtensionProject object,JI(`igrafxrf.HLP',`ExtensionProject_Object')}

ExtensionProjects Object

The ExtensionProjects object is a collection of ExtensionProject objects, and is only accessible from the Application object. From Visual Basic, Extension Projects can be added to the collection, loaded, unloaded, and removed from the collection.

The ExtensionProjects object provides the following functionality:

- The ability to access any ExtensionProject objects that have been created.
- The ability to determine how many ExtensionProject objects are in the collection.
- The ability to add an Extension Project to the collection.

Extension Projects cannot be created or deleted using Visual Basic. Creating a new Extension Project must be performed from the Tools menu in the iGrafx Professional user interface:

Tools->Visual Basic->Extension Projects…->New

Properties, Methods, and Events

All of the properties, methods, and events for the ExtensionProjects object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	Add	
Count	Item	
Parent		

Related Topics

Extension object
ExtensionProject object
IGrafxExtension object
iGrafx API Object Hierarchy

Add Method

Syntax *ExtensionProjects*.**Add**(*FileName* As String, [*VisibleInIDE* As Boolean = True], [*Load* As Boolean = True]) As ExtensionProject

Description The Add method adds an ExtensionProject object to the application's ExtensionProjects collection.

The *FileName* argument specifies the file name of an existing Extension Project file (.flx) on disk.

The *VisibleInIDE* argument specifies whether the ExtensionProject is visible in the Visual Basic Integrated Development Environment.

The *Load* argument specifies whether the ExtensionProject is Loaded as well as added. Refer to the ExtensionProject.Load method for more information.

Example The following example adds an Extension Project to the collection, and then displays the name of the project file. This example requires an Extension Project file on disk called "Sample.flx".

```
' Dimension the variables
Dim igxProject As ExtensionProject
' Add a project and get it's ExtensionProject object
Set igxProject = ExtensionProjects.Add("C:\iGrafx\Sample.flx")
' Display the name of the newly loaded project
MsgBox igxProject.FullName & " loaded."
```

See Also ExtensionProject.Load method

ExtensionProject.Remove method

{button ExtensionProjects object,JI(`igrafxrf.HLP',`ExtensionProjects_Object')}

Item Method

Syntax *ExtensionProjects*.**Item**(*Index* As Integer) As ExtensionProject

Description The Item method returns the ExtensionProject object at the specified *Index* from the ExtensionProjects collection. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type ExtensionProject.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is helpful to use error trapping before attempting to use the Item method.

Example The following example gathers the names of all the ExtensionProject objects into a string, and displays the names.

```
' Dimension the variables
Dim sString As String
' Iterate through the collection
For Index = 1 To ExtensionProjects.Count
   ' Gather the full names
   sString = sString & ExtensionProjects.Item(Index).FullName & Chr(13)
Next Index
' Display the names
MsgBox "Extension Projects: " & Chr(13) & Chr(13) & sString
```

{button ExtensionProjects object,JI(`igrafxrf.HLP',`ExtensionProjects_Object')}

StartPointNames Object

The StartPointNames object is a collection of strings that identifies all the start point names that have been assigned to shapes (the Shape.StartPointName property) within a diagram. Each diagram in a document has its own separate StartPointNames collection.

The StartPointNames object provides the following functionality:

- The ability to access any start point name string in the collection.
- The ability to determine how many start point names are in the collection.
- The ability to retrieve a Shape object based on searching the collection for its start point name.

Properties, Methods, and Events

All of the properties, methods, and events for the StartPointNames object are listed in the following table. Click the name to view the documentation for any property, method, or event.

Properties	Methods	Events
Application	FindStartPoint	
Count	Item	
Parent		

FindStartPoint Method

Syntax *StartPointNames*.**FindStartPoint**(*Name* As String) As Shape

Description The FindStartPoint method finds a Shape object based on its StartPointName property. The result of this method must be assigned to a variable of type Shape. The *Name* argument specifies the start point name to find.

A run-time error occurs if the *Name* argument is not found in the collection. The example code shows that the situation can be handled in two ways.

Example The following example creates two shapes and sets their StartPointName properties. Next, the FindStartPoint method is called to retrieve the Shape object with the designated start point name. In the example, an invalid start point is given, which generates a run-time error. (As alternatives, first run the code with the invalid name but without the error handler, then substitute a valid start point name.) The error handler displays a message and then code execution resumes at the next line. Here an alternative method of first searching the StartPointNames collection to find a match is used. If a match is found, then the FindStartPoint method is called to retrieve the Shape object.

```
' Dimension the variables
Dim igxShape As Shape
' Set igxShape variable to a new Shape object
With ActiveDiagram.DiagramObjects.AddShape(1440, 1440)
    ' Set the shape with a StartPointName
    .StartPointName = "StartShapeA"
    .Text = "StartShapeA"
End With
With ActiveDiagram.DiagramObjects.AddShape(1440 * 2, 1440 * 3)
    ' Set the shape with a StartPointName
    .StartPointName = "StartShapeB"
    .Text = "StartShapeB"
End With
' Use FindStartPoint method to get the shape that matches a start point name
On Error GoTo ErrorHandler
Set igxShape = ActiveDiagram.StartPointNames.FindStartPoint _
    ("StartShapeC")
' Find the Shape object that matches the StartPointName search
For iCount = 1 To ActiveDiagram.StartPointNames.Count
    If (ActiveDiagram.StartPointNames.Item(iCount) = "StartShapeB") Then
        MsgBox "StartPoint Name found in collection. Get and " _
            & "select the shape."
        Set igxShape = ActiveDiagram.StartPointNames.FindStartPoint _
            (ActiveDiagram.StartPointNames.Item(iCount))
        igxShape.DiagramObject.Selected = True
    End If
Next iCount
MsgBox "Found the StartPoint: " & igxShape.StartPointName

Exit Sub
ErrorHandler:
    MsgBox "Invalid Start Point Name"
    Resume Next
```

{button StartPointNames object,JI(`igrafxrf.HLP',`StartPointNames_Object')}

Item Method

Syntax *StartPointNames*.**Item**(*Index* As Integer) As String

Description The Item method returns the start point name string at the specified Index value. The data type of the *Index* argument is Integer. The result of the method must be assigned to a variable of type String.

Error If an invalid index value is supplied, then an Invalid Index Value error is returned. For this reason, it is helpful to use error trapping before attempting to use the Item method.

Example The following example creates a number of Shape obejcts and sets each StartPointName property. A message box displays the start point name and its position in the collection as each shape is created. Finally, the Item method is used to output all the start point names in the diagram to the Output window.

```
' Dimension the variables
Dim Index As Integer
' Create several new Shape objects
For Index = 1 To ((Rnd * 10) + 1)
    With ActiveDiagram.DiagramObjects.AddShape(1440 * Index, 1440 * Index)
        ' Give the shape a StartPointName
        .StartPointName = "MyStartShape" & Index
        MsgBox .StartPointName & "'s position in the collection " _
            & "is ActiveDiagram.StartPointNames.Item(" & Index & ")"
    End With
Next Index
' Display a list of StartPointNames in the Output window
For Index = 1 To ActiveDiagram.StartPointNames.Count
    Output ActiveDiagram.StartPointNames.Item(Index)
Next Index
```

{button StartPointNames object,JI(`igrafxrf.HLP',`StartPointNames_Object')}