Microsoft Visio Developer Reference

See also

Welcome to the Microsoft Visio Developer Reference documentation. This documentation includes the following:

What's new in Microsoft Visio 2002 for developers

Automation Reference

The topics in this section provide an overview of the Automation Reference, including information about the Visio object model, the Visio type library, extending the functionality of Visio with macros, add-ons, and COM (Component Object Model) add-ins, and a sample Microsoft Visual Basic for Applications (VBA) macro.

Look here for details on Visio objects, properties, methods, and events. This reference also includes information about common Automation tasks, such as starting the Visual Basic Editor; viewing Visio object, property, method, and event descriptions in the **Object Browser**; and running macros and add-ons.

Go to the Automation Reference

ShapeSheet Reference

The topics in this section provide an overview of the ShapeSheet Reference, including information about working with formulas, strings, date and time values, units of measure, and information about common ShapeSheet tasks, such as adding and deleting ShapeSheet sections, and referencing cells from formulas.

Look here for details on each section, row, and cell in a ShapeSheet spreadsheet and details on functions you can use in formulas.

Go to the ShapeSheet Reference

What's new in Microsoft Visio 2002 for developers

See also

Microsoft Visio 2002 provides a powerful single platform for your custom drawing solutions. New ShapeSheet cells and Automation objects, properties, methods, and events give you more options for defining the behavior of the elements in your solutions.

- New features in Visio 2002

The following topics provide lists of ShapeSheet and Automation elements that are new in Visio 2002.

New cells (alphabetic list)

New cells (by section)

New objects

New properties (alphabetic list)

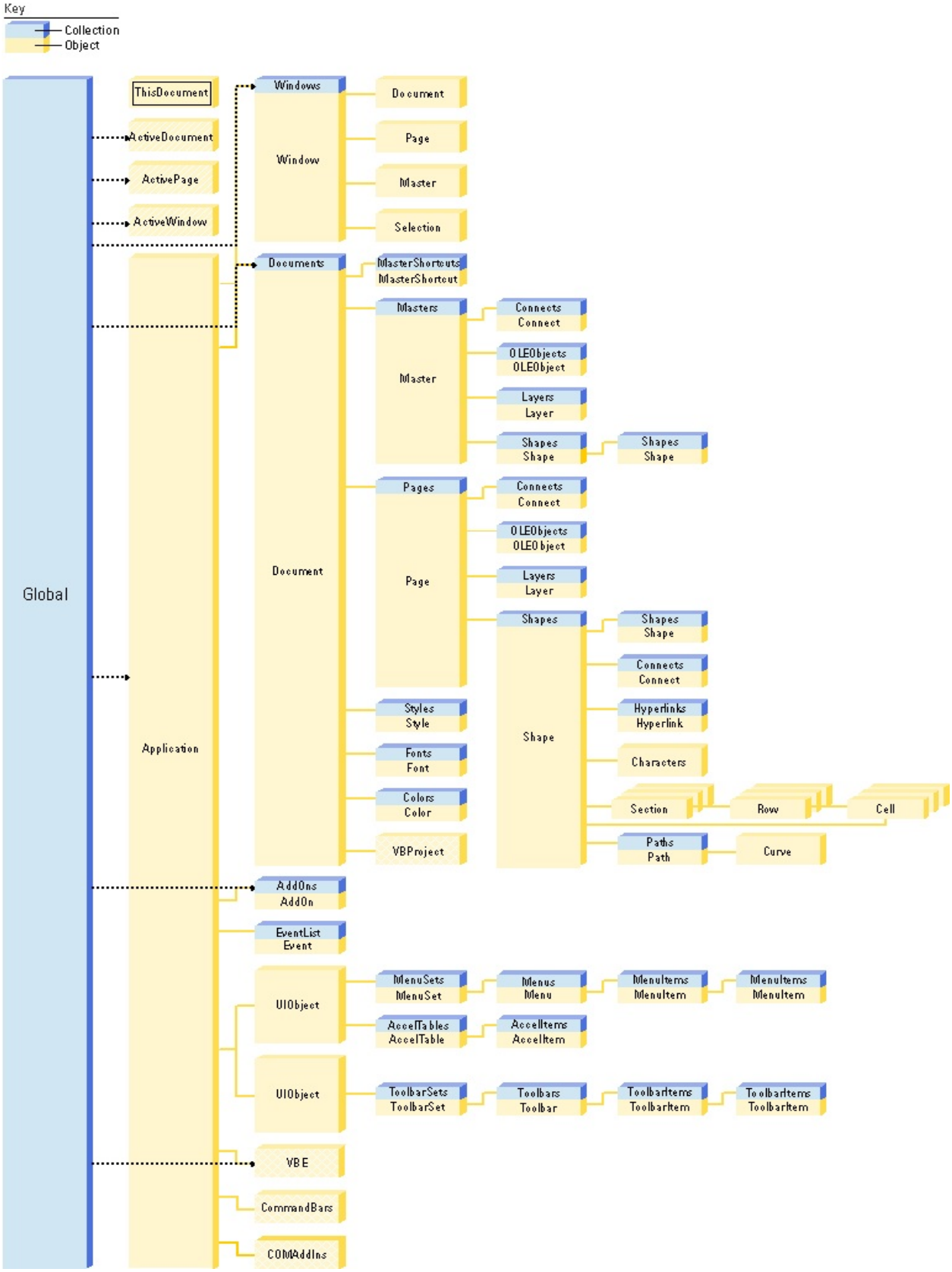
New properties (by object)

New methods (alphabetic list)

New methods (by object)

New events

The Microsoft® Visio® object model



About Automation

You can write programs to control Microsoft Visio in Microsoft Visual Basic for Applications (VBA), Microsoft Visual Basic, C++, or any programming language that supports Automation.

A program can use Automation to incorporate Visio drawing and diagramming capabilities or to automate simple repetitive tasks in Visio. For example, a program might generate an organization chart from a list of names and positions or print all of the masters on a stencil.

- How a program using Automation controls Visio
- The VBA programming environment in Visio

About extending the functionality of Microsoft Visio

You can extend the functionality of Microsoft Visio in the following ways:

Create Visio-specific macros and add-ons.

Create COM (Component Object Model) add-ins.

- Macros and add-ons
- COM add-ins

About XML in Microsoft Visio

Beginning in Microsoft Visio 2002, the Extensible Markup Language (XML) is supported as a native file format, making it possible to share your Visio drawings across multiple platforms.

Additionally, the Visio 2002 object model provides new properties and methods that make it possible for solution developers to store well-formed, solution-specific XML data within a Visio document.

For details about working with XML in Visio 2002, see the Microsoft Developer Network (MSDN) Web site.

About the Visio type library

Microsoft Visio products include a type library that defines the objects, properties, methods, events, and constants that Visio exposes to Automation clients. To use the Visio type library, a development environment must reference it. The Microsoft Visual Basic for Applications (VBA) project of a Visio document automatically references the Visio type library. In other development environments you must take appropriate steps to reference the library.

The names of the libraries your VBA project references are displayed in the **Project/Library** list in the **Object Browser** in the Visual Basic Editor.

Sample Microsoft VBA macro

For each drawing file that is open in the Microsoft Visio instance, the sample Microsoft Visual Basic for Applications (VBA) macro shown below does the following:

Logs the name and path of the drawing file in the Immediate window

Logs the name of each page in the Immediate window

Here is a look at the code in the program and what it does.

```
Public Sub ShowNames ()
    'Declare object variables as Visio object types
    Dim pagObj As Visio.Page
    Dim docObj As Visio.Document
    Dim docsObj As Visio.Documents
    Dim pagsObj As Visio.Pages
    'Iterate through all open documents
    Set docsObj = Application.Documents
    For Each docObj In docsObj
        'Print the drawing name in the Visual Basic Editor
        'Immediate Window
        Debug.Print docObj.FullName
```

```
        'Iterate through all pages in a drawing
        Set pagsObj = docObj.Pages
        For Each pagObj In pagsObj
            'Prints the page name in the Visual Basic Editor
            'Immediate Window
            Debug.Print Tab(5); pagObj.Name
        Next
    Next
End Sub
```

Here is an example of the program's output, assuming Office.vsd and Recycle.vsd are open and have been saved in the specified locations.

Sample output	Description
c:\visio\solutions\Office.vsd	The name of the first drawing
Background-1	The name of page 1
Background-2	The name of page 2
c:\visio\solutions\Recycle.vsd	The name of the second drawing
Page-1	The name of page 1
Page-2	The name of page 2

You can find more information about writing a program using the VBA environment and about the Visual Basic Editor in the Microsoft Visual Basic Help (in the Visual Basic Editor window, click **Microsoft Visual Basic Help** on the **Help** menu).

You can find details about using a specific Visio object, property, method, or event in the Automation Reference included in this Developer Reference (on the Visio **Help** menu, click **Developer Reference**).

Note If you did not install the Developer Reference at the time you installed Visio, clicking the **Developer Reference** command on the **Help** menu will automatically start its installation.

Add a macro or add-on to a shape's shortcut menu

Select the shape.

On the **Window** menu, click **Show ShapeSheet**.

Click in an Action cell in an Actions section.

> If you don't see an Actions section, insert one by clicking **Section** on the **Insert** menu, and then selecting the **Actions** check box in the **Insert Section** dialog box.

On the **Edit** menu, click **Action**.

In the **Action** dialog box, under **Properties**, enter the menu and prompt properties.

Click **Run macro**, and then select the program you want to run from the **Run macro** list. Click **OK**.

On the drawing page, right-click the shape, and then click the custom menu command on the shortcut menu to run the program.

Note You also can associate a macro or add-on with a shape by entering a formula that uses the RUNADDON function in any ShapeSheet cell.

Associate a macro or add-on with the double-click behavior of a shape

Select the shape with which you want to associate the macro or add-on.

On the **Format** menu, click **Behavior**.

In the **Behavior** dialog box, click the **Double-Click** tab, and then click **Run macro**.

From the **Run macro** list, select the macro or add-on you want to run.

Click **OK**.

Double-click the shape to run the macro or add-on.

Change the developer settings for Microsoft Visio

On the **Tools** menu, click **Options**.

In the **Options** dialog box, click the **Advanced** tab.

Under **Developer settings**, select the check boxes for the settings you want, and then click **OK**.

Run a macro or add-on from the Macros dialog box

In the Microsoft Visio window, click **Macros** on the **Tools** menu, and then click **Macros**.

Choose the macro or add-on you want to run, and then click **Run**.

Note If you have more than one Microsoft Visio drawing open and the macro you want to run does not appear in the **Macros** dialog box, make sure you have chosen the Visio document in which the macro is stored from the **Macros in** list.

Start the Visual Basic Editor

On the **Tools** menu, click **Macros**, and then click **Visual Basic Editor**. Or press ALT+F11.

Tip For quicker access to the Visual Basic Editor and other developer commands, you can show the **Developer** toolbar in Microsoft Visio. To do this, right-click anywhere in the Visio toolbar, and then click **Developer** on the shortcut menu.

View Visio object, property, method, event, and constant descriptions

In the Visual Basic Editor, click **Object Browser** on the **View** menu.

> The **Object Browser** initially displays items declared by all libraries referenced by your project.

In the **Project/Library** list, select **Visio**.

Note You also can view additional information about an item by selecting the item and pressing F1 for Help.

Event codes

See also

When you are working with the Add or AddAdvise method, use the following table to find the event code for the event you want to create. This table lists each Microsoft Visio event and its corresponding event code and numeric code.

Note If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), you don't need to create your own events. See the event topic in this reference that corresponds to the event you want to use.

- Table of events and corresponding event and numeric codes

AfterModal event

Example

Occurs after the Microsoft Visio instance leaves a modal state.

Version added

4.1

Syntax

```
Private Sub object_AfterModal(ByVal app As IVApplic
```

object The **WithEvents** object that receives the event.

Remarks

Visio becomes modal when it displays a dialog box. A modal instance of Visio does not handle Automation calls. The **BeforeModal** event indicates that the instance is about to become modal, and the **AfterModal** event indicates that the instance is no longer modal.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

AfterResume event

Example

Occurs when the operating system resumes normal operation after having been suspended.

Version added

2000 SR-1

Syntax

```
Private Sub object_AfterResume(ByVal app As IVAppl
```

object	The **WithEvents** object that receives the event.

Remarks

You can use the **AfterResume** event to reopen any network files that you may have closed in response to the **BeforeSuspend** event.

If your solution runs outside of the Visio process you cannot be assured of receiving this event. For this reason, you should monitor window messages in your program.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

AppActivated event

Example

Occurs after a Microsoft Visio instance becomes active.

Version added

4.1

Syntax

```
Private Sub object_AppActivated (ByVal app As IVApp
```

object The **WithEvents** object that receives the event.

Remarks

The **AppActivated** event indicates that an instance of Visio has become the active application on the Microsoft Windows desktop. The **AppActivated** event is different from the **AppObjectActivated** event, which occurs after an instance of Visio becomes active—the instance of Visio that is retrieved by the **GetObject** function in a Microsoft Visual Basic program.

If you're using Visual Basic or Visual Basic for Applications, the syntax in this

topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

AppDeactivated event

Example

Occurs after a Microsoft Visio instance becomes inactive.

Version added

4.1

Syntax

```
Private Sub object_AppDeactivated(ByVal app As IVA
```

object The **WithEvents** object that receives the event.

Remarks

The **AppDeactivated** event indicates that an instance of Visio is no longer the active application on the Microsoft Windows desktop. The **AppDeactivated** event is different from the **AppObjectDeactivated** event, which occurs after an instance of Visio ceases to be the active instance—the instance of Visio that is retrieved by the **GetObject** function in a Microsoft Visual Basic program.

If you're using Visual Basic or Visual Basic for Applications, the syntax in this

topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

AppObjActivated event

Example

Occurs after a Microsoft Visio instance becomes active.

Version added

4.1

Syntax

```
Private Sub object_AppObjActivated(ByVal app As IV
```

object The **WithEvents** object that receives the event.

Remarks

The **AppObjActivated** event indicates that an instance of Visio has become active—the instance of Visio that is retrieved by the **GetObject** function in a Microsoft Visual Basic program. The **AppObjActivated** event is different from the **AppActivated** event, which occurs after an instance of Visio becomes the active application on the Microsoft Windows desktop.

If you're using Visual Basic or Visual Basic for Applications, the syntax in this

topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

AppObjDeactivated event

Example

Occurs after a Microsoft Visio instance becomes inactive.

Version added

4.1

Syntax

```
Private Sub object_AppObjDeactivated(ByVal app As I
```

object	The **WithEvents** object that receives the event.

Remarks

The **AppObjDeactivated** event indicates that the instance of Visio is no longer the active instance—the instance of Visio that is retrieved by the **GetObject** function in a Microsoft Visual Basic program. The **AppObjDeactivated** event is different from the **AppDeactivated** event, which occurs after an instance of Visio becomes inactive.

If you're using Visual Basic or Visual Basic for Applications, the syntax in this

topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

BeforeDocumentClose event

Example

Occurs before a document is closed.

Version added

4.1

Syntax

```
Private Sub object_BeforeDocumentClose(ByVal doc A
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

BeforeDocumentSave event

Example

Occurs just before a document is saved.

Version added

5.0

Syntax

```
Private Sub object_BeforeDocumentSave(ByVal doc As
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

BeforeDocumentSaveAs event

Example

Occurs just before a document is saved using the **Save As** command.

Version added

5.0

Syntax

```
Private Sub object_BeforeDocumentSaveAs(ByVal doc
```

object The **WithEvents** object that receives the event.

Remarks

The **BeforeDocumentSaveAs** event fires when saving to either a native format (for example, VSD or VDX) or a non-native format (for example, HTM or BMP). It does not fire when saving to DWG, DXF, and DGN formats.

The **BeforeDocumentSaveAs** event is one of a group of events for which the **EventInfo** property of the **Application** object contains extra information.

If the **BeforeDocumentSaveAs** event is fired because a save was initiated by a user or a program, the **EventInfo** property returns the following string:

"/saveasfile=<filename>"

If it fires because Visio is saving a copy of an open file (for autorecovery or to include as a mail attachment), the **EventInfo** property will return one of the following strings:

If the event is fired for autorecovery purposes, the name of a recovery file in this format: "/autosavefile=C:\TEMP\~$2VSO2FD.vsd"

If the event is fired because a document copy is being made to send as a mail attachment, the name of an attachment file in this format: "/mailfile=C:\TEMP\~$2VSO2FD.vsd"

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

If you are handling this event from a program that receives a notification over a connection using the **AddAdvise** method, the *vMoreInfo* argument to **VisEventProc** designates the document index: "/doc=1".

BeforeMasterDelete event

Example

Occurs before a master is deleted from a document.

Version added

4.1

Syntax

```
Private Sub object_BeforeMasterDelete(ByVal Master A
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

BeforeModal event

Example

Occurs before a Microsoft Visio instance enters a modal state.

Version added

4.1

Syntax

```
Private Sub object_BeforeModal(ByVal app As IVAppli
```

object The **WithEvents** object that receives the event.

Remarks

Visio becomes modal when it displays a dialog box. A modal instance of Visio does not handle Automation calls. The **BeforeModal** event indicates that an instance is about to become modal, and the **AfterModal** event indicates that the instance is no longer modal.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

BeforePageDelete event

Example

Occurs before a page is deleted.

Version added

4.1

Syntax

```
Private Sub object_BeforePageDelete(ByVal Page As IV
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

BeforeQuit event

Example

Occurs before a Microsoft Visio instance terminates.

Version added

4.1

Syntax

```
Private Sub object_BeforeQuit(ByVal app As IVApplica
```

object The **WithEvents** object that receives the event.

Remarks

When programming with Microsoft Visual Basic for Applications (VBA), use the **BeforeDocumentClose** event instead of the **BeforeQuit** event. The code in a VBA project of a Visio document never has the chance to respond to the **BeforeQuit** event because the project is a property of a document, and all documents are closed before the **BeforeQuit** event notification is sent.

If you're using Microsoft Visual Basic or VBA, the syntax in this topic describes

a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

BeforeSelectionDelete event

Example

Occurs before selected objects are deleted.

Version added

4.1

Syntax

```
Private Sub object_BeforeSelectionDelete(ByVal Select
```

object	The **WithEvents** object that receives the event.

Remarks

A **Shape** object can serve as the source object for the **BeforeSelectionDelete** event if the shape's **Type** property is **visTypeGroup**(2) or **visTypePage**(1).

The **BeforeSelectionDelete** event indicates that selected shapes are about to be deleted. This notification is sent whether or not any of the shapes are locked; however, locked shapes aren't deleted. To find out if a shape is locked against deletion, check the value of its LockDelete cell.

The **BeforeSelectionDelete** and **BeforeShapeDelete** events are similar in that they both fire before shape(s) are deleted. They differ in how they behave when a single operation deletes several shapes. Suppose a **Cut** operation deletes three shapes. The **BeforeShapeDelete** event fires three times and acts on each of the three objects. The **BeforeSelectionDelete** event fires once and it acts on a **Selection** object in which the three shapes that you want to delete are selected.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

BeforeShapeDelete event

Example

Occurs before a shape is deleted.

Version added

4.5

Syntax

```
Private Sub object_BeforeShapeDelete(ByVal Shape As
```

object	The **WithEvents** object that receives the event.

Remarks

A **Shape** object can serve as the source object for the **BeforeShapeDelete** event if the shape's **Type** property is **visTypeGroup**(2) or **visTypePage**(1).

The **BeforeSelectionDelete** and **BeforeShapeDelete** events are similar in that they both fire before shape(s) are deleted. They differ in how they behave when

a single operation deletes several shapes. Suppose a **Cut** operation deletes three shapes. The **BeforeShapeDelete** event fires three times and acts on each of the three objects. The **BeforeSelectionDelete** event fires once and it acts on a **Selection** object in which the three shapes that you want to delete are selected.

If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

Note The **BeforeShapeDelete** event is included in the event set of all the objects in the **Applies to** list. For those objects you can use VBA **Dim WithEvents** variables to sink the **BeforeShapeDelete** event.

For performance considerations, the **Document** object's event set does not include the **BeforeShapeDelete** event. To sink the **BeforeShapeDelete** event from a **Document** object (and the **ThisDocument** object in a VBA project), you must use the **AddAdvise** method.

BeforeShapeTextEdit event

Example

Occurs before a shape is opened for text editing in the user interface.

Version added

2000

Syntax

```
Private Sub object_BeforeShapeTextEdit(ByVal Shape
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

BeforeStyleDelete event

Example

Occurs before a style is deleted.

Version added

4.1

Syntax

```
Private Sub object_BeforeStyleDelete(ByVal Style As I
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

BeforeSuspend event

Example

Occurs before the operating system enters a suspended state.

Version added

2000 SR-1

Syntax

```
Private Sub object_BeforeSuspend(ByVal app As IVapp
```

object The **WithEvents** object that receives the event.

Remarks

Client programs should close any open network files when this event is fired.

If your solution runs outside of the Visio process you cannot be assured of receiving this event. For this reason, you should monitor window messages in your program.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the

syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

BeforeWindowClosed event

Example

Occurs before a window is closed.

Version added

4.1

Syntax

```
Private Sub object_BeforeWindowClosed(ByVal Windo
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

BeforeWindowPageTurn event

Example

Occurs before a window is about to show a different page.

Version added

4.5

Syntax

```
Private Sub object_BeforeWindowPageTurn(ByVal Wi
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

BeforeWindowSelDelete event

Example

Occurs before the shapes in the selection of a window are deleted.

Version added

4.1

Syntax

```
Private Sub object_BeforeWindowSelDelete(ByVal Wi
```

object	The **WithEvents** object that receives the event.

Remarks

The **BeforeWindowSelDelete** event fires if user interactions cause shapes in a window to be deleted. It doesn't fire if a program deletes shapes in a window using the **Cut** method, for example.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

CellChanged event

Occurs after the value changes in a cell in a document.

Version added

4.1

Syntax

```
Private Sub object_CellChanged(ByVal Cell As IVCell)
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

Note The **CellChanged** event is included in the event set of all the objects in the **Applies to** list. For those objects you can use VBA **Dim WithEvents** variables to sink the **CellChanged** event.

For performance considerations, the **Document** object's event set does not include the **CellChanged** event. To sink the **CellChanged** event from a **Document** object (and the **ThisDocument** object in a VBA project), you must use the **AddAdvise** method.

Example

This class module demonstrates defining a sink class called ShapeSink that declares the object variable *m_shpObj* using the **WithEvents** keyword. It contains a procedure, **InitWith**, that assigns a particular **Shape** object, *aShape*, to *m_shpObj*. The class module also contains an event handler for the **CellChanged** event, which can be fired by a **Shape** object—in this case, the **Shape** object represented by *aShape*.

```
Dim WithEvents m_shpObj As Visio.Shape

Public Sub InitWith(ByVal aShape As Visio.Shape)
    Set m_shpObj = aShape
End Sub

Private Sub m_shpObj_CellChanged(ByVal Cell As Visio
    Debug.Print Cell.Shape.Name & " " & Cell.Name & " c
End Sub
```

ConnectionsAdded event

Example

Occurs after connections have been established between shapes.

Version added

5.0

Syntax

```
Private Sub object_ConnectionsAdded(ByVal Connects
```

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it

applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

Note The **ConnectionsAdded** event is included in the event set of all the objects in the **Applies to** list. For those objects you can use VBA **Dim WithEvents** variables to sink the **ConnectionsAdded** event.

For performance considerations, the **Document** object's event set does not include the **ConnectionsAdded** event. To sink the **ConnectionsAdded** event from a **Document** object (and the **ThisDocument** object in a VBA project), you must use the **AddAdvise** method.

ConnectionsDeleted event

Example

Occurs after connections between shapes have been removed.

Version added

5.0

Syntax

```
Private Sub object_ConnectionsDeleted(ByVal Connect
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

Note The **ConnectionsDeleted** event is included in the event set of all the objects in the **Applies to** list. For those objects you can use VBA **Dim WithEvents** variables to sink the **ConnectionsDeleted** event.

For performance considerations, the **Document** object's event set does not include the **ConnectionsDeleted** event. To sink the **ConnectionsDeleted** event from a **Document** object (and the **ThisDocument** object in a VBA project), you must use the **AddAdvise** method.

ConvertToGroupCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelConvertToGroup** event.

Version added

2000

Syntax

Private Sub *object*_**ConvertToGroupCanceled**(**ByVal** S

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives

notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

DesignModeEntered event

Example

Occurs before a document enters design mode.

Version added

5.0

Syntax

```
Private Sub object_DesignModeEntered(ByVal doc As
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

DocumentAdded event

Example

Occurs after a document is opened or created.

Version added

4.1

Syntax

You can only handle the **DocumentAdded** event by creating an **Event** object using the **Add** or **AddAdvise** method. See those method topics for details about the correct syntax.

Remarks

You can add **DocumentAdded** events to the **EventList** property of **Application**, **Documents**, or **Document** objects. The first two are straightforward—if a document is opened or created in the scope of the **Application** object or its **Documents** collection, the **DocumentAdded** event occurs.

However, adding a **DocumentAdded** event to the **EventList** property of a **Document** object makes sense only if the event's action is

visActCodeRunAddon. In this case, the event is persistable—it can be stored with the document. If the document that contains the persistent event is opened, its action is triggered. If a new document is based on or copied from the document that contains the persistent event, the **DocumentAdded** event is copied to the new document and its action is triggered. However, if the event's action is **visActCodeAdvise**, that event is not persistable and therefore is not stored with the document; hence, it is never triggered.

You can prevent code from running in response to the **DocumentCreated**, **DocumentOpened**, or **DocumentAdded** event and all events from firing by setting the value of the **EventsEnabled** property of an **Application** object to **False**, or by adding the entry, EventsEnabled=0, to the Visio Application section in the registry.

DocumentChanged event

Example

Occurs after certain properties of a document are changed.

Version added

4.1

Syntax

```
Private Sub object_DocumentChanged(ByVal doc As I
```

object The **WithEvents** object that receives the event.

Remarks

The **DocumentChanged** event indicates that one of a document's properties, such as **Author** or **Description**, has changed.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise**

method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

DocumentCloseCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelDocumentClose** event.

Version added

2000

Syntax

Private Sub *object*_**DocumentCloseCanceled(ByVal** do

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives

notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

DocumentCreated event

Occurs after a document is created.

Version added

4.1

Syntax

```
Private Sub object_DocumentCreated(ByVal doc As IV
```

object The **WithEvents** object that receives the event.

Remarks

The **DocumentCreated** event is often added to the **EventList** collection of a Visio template file (.vst). The event's action is triggered whenever a new document is created based on that template.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

You can add **DocumentCreated** events to the **EventList** collection of an **Application** object, **Documents** collection, or **Document** object. The first two are straightforward—if a document is opened or created in the scope of the **Application** object or its **Documents** collection, the **DocumentCreated** event occurs.

However, adding a **DocumentCreated** event to the **EventList** collection of a **Document** object makes sense only if the event's action is **visActCodeRunAddon**. In this case, the event is persistable—it can be stored with the document. If the document that contains the persistent event is opened, its action is triggered. If a new document is based on or copied from the document that contains the persistent event, the **DocumentCreated** event is copied to the new document and its action is triggered. However, if the event's action is **visActCodeAdvise**, that event is not persistable and therefore is not stored with the document; hence it is never triggered.

You can prevent code from running in response to the **DocumentCreated**, **DocumentOpened**, or **DocumentAdded** event and all events from firing by setting the value of the **EventsEnabled** property of an **Application** object to **False**, or by adding the entry, EventsEnabled=0 to the Visio Application section in the registry.

DocumentOpened event

Example

Occurs after a document is opened.

Version added

4.1

Syntax

```
Private Sub object_DocumentOpened(ByVal doc As IV
```

object The **WithEvents** object that receives the event.

Remarks

The **DocumentOpened** event is often added to the **EventList** collection of a Visio template file (.vst). The event's action is triggered whenever an existing document is opened.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

You can add **DocumentOpened** events to the **EventList** collection of an **Application** object, **Documents** collection, or **Document** object. The first two are straightforward—if a document is opened or created in the scope of the **Application** object or its **Documents** collection, the **DocumentOpened** event occurs.

However, adding a **DocumentOpened** event to the **EventList** collection of a **Document** object makes sense only if the event's action is **visActCodeRunAddon**. In this case, the event is persistable—it can be stored with the document. If the document that contains the persistent event is opened, its action is triggered. If a new document is based on or copied from the document that contains the persistent event, the **DocumentOpened** event is copied to the new document and its action is triggered. However, if the event's action is **visActCodeAdvise**, that event is not persistable and therefore is not stored with the document; hence it is never triggered.

You can prevent code from running in response to the **DocumentCreated**, **DocumentOpened** or **DocumentAdded** event and all events from firing by setting the value of the **EventsEnabled** property of an **Application** object to **False**, or by adding the entry, EventsEnabled=0, to the Visio Application section in the registry.

DocumentSaved event

Occurs after a document is saved.

Version added

4.1

Syntax

```
Private Sub object_DocumentSaved(ByVal doc As IVD
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

DocumentSavedAs event

Example

Occurs after a document is saved using the **Save As** command.

Version added

4.1

Syntax

```
Private Sub object_DocumentSavedAs(ByVal doc As IV
```

object The **WithEvents** object that receives the event.

Remarks

The **DocumentSavedAs** event is one of a group of events for which the **EventInfo** property of the **Application** object contains extra information.

If the **DocumentSavedAs** event is fired because a save was initiated by a user or a program, the **EventInfo** property returns the following string:

"/saveasfile=<filename>"

If it fires because Visio is saving a copy of an open file (for autorecovery or to include as a mail attachment), the **EventInfo** property returns one of the following strings:

If the event is fired for autorecovery purposes, the name of a recovery file in this format: "/autosavefile=C:\TEMP\~$2VSO2FD.vsd"

If the event is fired because a document copy is being made to send as a mail attachment, the name of an attachment file in this format: "/mailfile=C:\TEMP\~$2VSO2FD.vsd"

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

If you are handling this event from a program that receives a notification over a connection created with the **AddAdvise** method, the vMoreInfo argument to **VisEventProc** designates the document index: "/doc=1".

EnterScope event

Queued when an internal command begins, or when an Automation client opens a scope using the **BeginUndoScope** method.

Version added

2000

Syntax

Private Sub Application_EnterScope (ByVal Applicatic

nScopeID	A language independent number that
bstrDescription	A textual description of the operation description passed to the **BeginUndo**

Remarks

The *nScopeID* value returned in the case of a Visio operation is the equivalent of the command related constants that begin with **visCmd***.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the

syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

If you are handling this event from a program that receives a notification over a connection created with the **AddAdvise** method, the **EnterScope** event is one of a group of selected events that record extra information in the **EventInfo** property of the **Application** object.

The **EventInfo** property returns *bstrDescription*, as described above. In addition, the **vMoreInfo** argument to **VisEventProc** will contain a string formatted as follows: [<nScopeID>;<bErrOrCancelled>;<bstrDescription>]

For **EnterScope**, *bErrOrCancelled* will always equal zero.

ExitScope event

Queued when an internal command ends, or when an Automation client exits a scope using the **EndUndoScope** method.

Version added

2000

Syntax

```
Private Sub Application_ExitScope (ByVal Application
```

nScopeID	A language independent number that describes the operati
bstrDescription	A textual description of the operation that changes in diffe
bErrOrCancelled	**True** if there was an error during the scope or if the scope

Remarks

The *nScopeID* value returned in the case of a Visio operation is the equivalent of the command related constants that begin with **visCmd***.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the

syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

If you are handling this event from a program that receives a notification over a connection created using the **AddAdvise method**, the **ExitScope** event is one of a group of selected events that record extra information in the **EventInfo** property of the **Application** object.

The **EventInfo** property returns *bstrDescription*, as described above. In addition, the **vMoreInfo** argument to **VisEventProc** will contain a string formatted as follows: [<nScopeID>;<bErrOrCancelled>;<bstrDescription>]

For **ExitScope**, *bErrOrCancelled* will be non-zero if the operation failed or was canceled.

FormulaChanged event

Example

Occurs after a formula changes in a cell in a document.

Version added

5.0

Syntax

```
Private Sub object_FormulaChanged(ByVal Cell As IV
```

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it

applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

Note The **FormulaChanged** event is included in the event set of all the objects in the **Applies to** list. For those objects you can use VBA **Dim WithEvents** variables to sink the **FormulaChanged** event.

For performance considerations, the **Document** object's event set does not include the **FormulaChanged** event. To sink the **FormulaChanged** event from a **Document** object (and the **ThisDocument** object in a VBA project), you must use the **AddAdvise** method.

MarkerEvent event

Caused by invoking the **QueueMarkerEvent** method.

Version added

5.0

Syntax

```
Private Sub object_MarkerEvent(ByVal app As IVAppl
```

object	The **WithEvents** object that receives the event.
SequenceNum	The ordinal position of this event with respect to past events.
ContextString	Context string passed by the **QueueMarkerEvent** method.

Remarks

Unlike other events that Visio fires, a client program causes the **MarkerEvent** event to fire. A client program receives the **MarkerEvent** event only if the client program invoked the **QueueMarkerEvent** method.

By using the **MarkerEvent** event in conjunction with the **QueueMarkerEvent**

method, a client program can queue an event to itself. The client program receives the **MarkerEvent** event after Visio fires all the events present in its event queue at the time of the **QueueMarkerEvent** call.

The **MarkerEvent** event passes both the context string that was passed by the **QueueMarkerEvent** method and the sequence number of the **MarkerEvent** event to the **MarkerEvent** event handler. Either of these values can be used to correlate **QueueMarkerEvent** calls with **MarkerEvent** events. In this way, a client program can distinguish events it caused from those it did not cause.

For example, a client program that changes the values of Visio cells may only want to respond to the **CellChanged** events that it did not cause. The client program can first call the **QueueMarkerEvent** method and pass a context string for later use to bracket the scope of its processing:

```
visObj.QueueMarkerEvent "ScopeStart"
   <My program changes cells here>
visObj.QueueMarkerEvent "ScopeEnd"
```

Then, in the **MarkerEvent** event handler, the client program could use the context string passed to the **QueueMarkerEvent** method to identify the **CellChanged** events that it caused:

```
Dim ICausedCellChanges as Boolean

Private Sub visObj_MarkerEvent (ByVal App As Visio.IV
   ByVal SequenceNum As Long, ByVal ContextString As
   If ContextString = "ScopeStart" Then
      ICausedCellChanges = True
   ElseIf ContextString = "ScopeEnd" Then
      ICausedCellChanges = "False"
   End If
End Sub
```

```
Private Sub visObj_CellChanged (ByVal Cell As Visio.IV
    'Respond only if this client didn't cause cell change
    If ICausedCellChanges = False Then
        <respond to the cell changes>
    End If
End Sub
```

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

If you are handling this event from a program that receives a notification, the **MarkerEvent** event is one of one of a group of events that record extra information in the **EventInfo** property of the **Application** object.

The **EventInfo** property returns *contextstring* as described above. The **vMoreInfo** argument to **VisEventProc** will be empty.

MasterAdded event

Example

Occurs after a new master is added to a document.

Version added

4.1

Syntax

```
Private Sub object_MasterAdded(ByVal Master As IVM
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

MasterChanged event

Example

Occurs after properties of a master are changed and propagated to its instances.

Version added

4.1

Syntax

```
Private Sub object_MasterChanged(ByVal Master As I
```

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it

applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

MasterDeleteCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelMasterDelete** event.

Version added

2000

Syntax

```
Private Sub object_MasterDeleteCanceled(ByVal Mast
```

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Microsoft Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it

applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

MustFlushScopeBeginning event

Example

Occurs before the Microsoft Visio instance is forced to flush its event queue.

Version added

5.0

Syntax

```
Private Sub object_MustFlushScopeBeginning(ByVal a
```

object	The **WithEvents** object that receives the event.

Remarks

This event, along with the **MustFlushScopeEnded** event, can be used to identify whether or not an event is being fired because Visio is forced to flush its event queue.

Visio maintains a queue of pending events that it attempts to fire at discrete

moments when it is able to process arbitrary requests (callbacks) from event handlers.

Occasionally, Visio is forced to flush its event queue when it is not prepared to handle arbitrary requests. When this occurs, Visio first fires a **MustFlushScopeBeginning** event, then it fires the events that are presently in its event queue. After firing all pending events, Visio fires the **MustFlushScopeEnded** event.

After Visio has fired the **MustFlushScopeBeginning** event, client programs should not invoke Visio methods that have side effects until the **MustFlushScopeEnded** event is received. A client can perform arbitrary queries of Visio objects when Visio is between the **MustFlushScopeBeginning** event and **MustFlushScopeEnded** event, but operations that cause side effects may fail.

Visio performs a forced flush of its event queue immediately prior to firing a before event such as **BeforeDocumentClose** or **BeforeShapeDelete** because queued events may apply to objects that are about to close or be deleted. Using the **BeforeDocumentClose** event as an example, there can be queued events that apply to a shape object in the document that is being closed. So, before the document closes, Visio fires all the events in its event queue.

Events are fired in the following sequence when a shape is deleted:

MustFlushScopeBeginning event
Client should not invoke methods with side effects.

There are zero (0) or more events in the event queue.

BeforeShapeDelete event
Shape viable, but Visio is going to delete it.

MustFlushScopeEnded event
Client can resume invoking methods with side effects.

ShapesDeleted event
Shape has been deleted.

NoEventsPending event

No events remain to be fired.

An event is fired both before (**BeforeShapeDeleted** event) and after (**ShapesDeleted** event) the shape is deleted. If a program monitoring these events requires that additional shapes be deleted in response to the initial shape delete, it should do so in the **ShapesDeleted** event handler, not the **BeforeShapeDeleted** event handler. The **BeforeShapeDeleted** event is inside the scope of the **MustFlushScopeBeginning** event and the **MustFlushScopeEnded** event, while the **ShapesDeleted** event is not.

Note The sequence number of a **MustFlushScopeBeginning** event may be higher than the sequence number of events the client sees after it has received the **MustFlushScopeBeginning** event because Visio assigns sequence numbers to events as they occur. Any events that were queued when the forced flush began have a lower sequence number than the **MustFlushScopeBeginning** event, even though the **MustFlushScopeBeginning** event fires first.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

MustFlushScopeEnded event

Example

Occurs after the Microsoft Visio instance forces a flush of its event queue.

Version added

5.0

Syntax

```
Private Sub object_MustFlushScopeEnded(ByVal app A
```

object	The **WithEvents** object that receives the event.

Remarks

This event, along with the **MustFlushScopeBeginning** event, can be used to identify whether or not an event is being fired because Visio is forced to flush its event queue.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

NoEventsPending event

Example

Occurs after the Microsoft Visio instance flushes its event queue.

Version added

5.0

Syntax

```
Private Sub object_NoEventsPending(ByVal app As IV
```

object	The **WithEvents** object that receives the event.

Remarks

Visio maintains a queue of events and fires them at discrete moments. Immediately after Visio fires the last event in its event queue, it fires a **NoEventsPending** event.

A client program can use the **NoEventsPending** event as a signal that Visio has completed a burst of activity. For example, a client program may want to react to changes in a shape's geometry. A single user action performed on the shape can

generate several **CellChanged** events. The client program could record selected information for each **CellChanged** event and perform its processing after it receives the **NoEventsPending** event.

Visio fires the **NoEventsPending** event only if at least one of the events in the queue is being listened to. If no program is listening for any of the queued events, the **NoEventsPending** event does not fire. If your program is only listening to the **NoEventsPending** event, it does not fire unless another program is listening for some of the queued events.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

OnKeystrokeMessageForAddon event

Example

Occurs when Microsoft Visio receives a keystroke message from Microsoft Windows that is targeted at an add-on window or child of an add-on window.

Version added

2002

Syntax

Private Sub *object*_**OnKeystrokeMessageForAddon(By**

object	The **WithEvents** object that receives the event.
return value	**True** to indicate that the message was handled by the

Remarks

The **OnKeystrokeMessageForAddon** event enables add-ons to intercept and process accelerator and keystroke messages directed at their own add-on windows and child windows of their add-on windows. Only add-on windows

created using the **Add** method will source this event.

For this event to fire, the add-on window or one of its child windows must have keystroke focus and the Visio message loop must receive the keystroke message. It does not fire if the message loop associated with an add-on is handling messages instead of Visio.

Visio fires the **OnKeystrokeMessageForAddon** event when it receives messages in the following range:

WM_KEYDOWN	0x0100
WM_KEYUP	0x0101
WM_CHAR	0x0102
WM_DEADCHAR	0x0103
WM_SYSKEYDOWN	0x0104
WM_SYSKEYUP	0x0105
WM_SYSCHAR	0x0106
WM_SYSDEADCHAR	0x0107

The **MSGWrap** object, passed to the event handler when the **OnKeystrokeMessageForAddon** event fires, wraps the Windows **MSG** structure, which contains message data. See the **MSGWrap** object for more information, or refer to your Windows documentation.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

PageAdded event

Occurs after a new page is added to a document.

Version added

4.1

Syntax

```
Private Sub object_PageAdded(ByVal Page As IVPage)
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

PageChanged event

Example

Occurs after the name of a page, the background page associated with a page, or the page type (foreground or background) changes.

Version added

4.1

Syntax

```
Private Sub object_PageChanged(ByVal Page As IVPag
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives

notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

PageDeleteCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelPageDelete** event.

Version added

2000

Syntax

```
Private Sub object_PageDeleteCanceled(ByVal Page As
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Microsoft Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it

applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelConvertToGroup event

Example

Occurs before the application converts a selection of shapes to a group in response to a user action in the interface. If any event handler returns **True**, the operation is canceled.

Version added

2000

Syntax

Private Function *object*_**QueryCancelConvertToGroup**

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the operation.

Remarks

A Visio instance fires **QueryCancelConvertToGroup** after the user has directed the instance to convert one or more shapes into groups.

If any event handler returns **True** (cancel), the instance will fire

ConvertToGroupCanceled and not convert the shapes.

If all handlers return **False** (don't cancel), the conversion will be performed.

In some cases, such as when a shape with a **ForeignType** property of **visTypeMetafile** is converted to a group, the initial shape will be deleted and replaced with new shapes. In such cases the Visio instance will subsequently fire **BeforeSelectionDelete** and **BeforeShapeDelete** events before converting the shapes.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelDocumentClose event

Example

Occurs before the application closes a document in response to a user action in the interface. If any event handler returns **True**, the operation is canceled.

Version added

2000

Syntax

Private Function *object*_**QueryCancelDocumentClose(**

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the oper

Remarks

A Visio instance fires **QueryCancelDocumentClose** after the user has directed the instance to close a document.

If any event handler returns **True** (cancel), the instance will fire **DocumentCloseCanceled** and not close the document.

If all handlers return **False** (don't cancel), the instance will fire **BeforeDocumentClose** and then close the document.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelMasterDelete event

Example

Occurs before the application deletes a master in response to a user action in the interface. If any event handler returns **True**, the operation is canceled.

Version added

2000

Syntax

Private Function *object*_QueryCancelMasterDelete(By

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the opera

Remarks

A Visio instance fires **QueryCancelMasterDelete** after the user has directed the instance to delete a master.

If any event handler returns **True** (cancel), the instance will fire **MasterDeleteCanceled** and not delete the master.

If all handlers return **False** (don't cancel), the instance will fire **BeforeMasterDelete** and then delete the master.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelPageDelete event

Example

Occurs before the application deletes a page in response to a user action in the interface. If any event handler returns **True**, the operation is canceled.

Version added

2000

Syntax

Private Function *object*_**QueryCancelPageDelete(ByVa**

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the operati

Remarks

A Visio instance fires **QueryCancelPageDelete** after the user has directed the instance to delete a page.

If any event handler returns **True** (cancel), the instance will fire **PageDeleteCanceled** and not delete the page.

If all handlers return **False** (don't cancel) the instance will fire **BeforePageDelete**, and then delete the page.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelQuit event

Example

Occurs before the application terminates in response to a user action in the interface. If any event handler returns **True**, the operation is canceled.

Version added

2000

Syntax

Private Function *object*_**QueryCancelQuit**(**ByVal** app A

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the operatic

Remarks

A Visio instance fires **QueryCancelQuit** after the user has directed the instance to terminate.

If any event handler returns **True** (cancel), the instance will fire **QuitCanceled** and not terminate.

If all handlers return **False** (don't cancel), the instance will fire **BeforeQuit** and then terminate.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelSelectionDelete event

Example

Occurs before the application deletes a selection of shapes in response to a user action in the interface. If any event handler returns **True**, the operation is canceled.

Version added

2000

Syntax

Private Function *object*_**QueryCancelSelectionDelete(**

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the operation.

Remarks

A Visio instance fires **QueryCancelSelectionDelete** after the user has directed the instance to delete one or more shapes.

If any event handler returns **True** (cancel), the instance will fire

SelectionDeleteCanceled and not delete the shapes.

If all handlers return **False** (don't cancel), the instance will fire **BeforeSelectionDelete** and **BeforeShapeDelete**, and then delete the shapes.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelStyleDelete event

Example

Occurs before the application deletes a style in response to a user action in the interface. If any event handler returns **True**, the operation is canceled.

Version added

2000

Syntax

Private Function *object*_**QueryCancelStyleDelete(ByVa**

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the operati

Remarks

A Visio instance fires **QueryCancelStyleDelete** after the user has directed the instance to delete a style.

If any event handler returns **True** (cancel), the instance will fire **StyleDeleteCanceled** and not delete the style.

If all handlers return **False** (don't cancel), the instance will fire **BeforeStyleDelete** and then delete the style.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelSuspend event

Occurs before the operating system enters a suspended state. If any event handler returns **True**, the Microsoft Visio instance will deny the operating system's request.

Version added

2000 SR-1

Syntax

Private Function *object*_**QueryCancelSuspend(ByVal a**

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the operati

Remarks

You will typically respond **False** and allow the operating system to enter a suspended state. If you have open network files you can close them when you receive the **BeforeSuspend** event. If you have open network files that you cannot close you can return **True** and Visio will deny the operating system's

request.

If any event handler returns **True** (cancel), the instance will fire **SuspendCanceled** and not enter a suspended state.

If all handlers return **False** (don't cancel), the instance will fire **BeforeSuspend** and then enter a suspended state.

If your solution runs outside of the Visio process you cannot be assured of receiving this event. For this reason, you should monitor window messages in your program.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelUngroup event

Example

Occurs before the application ungroups a selection of shapes in response to a user action in the interface. If any event handler returns **True**, the operation is canceled.

Version added

2000

Syntax

Private Function *object*_**QueryCancelUngroup(ByVal**

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the opera

Remarks

A Visio instance fires **QueryCancelUngroup** after the user has directed the instance to ungroup one or more shapes.

If any event handler returns **True** (cancel), the instance will fire

UngroupCanceled and not ungroup the shapes.

If all handlers return **False** (don't cancel), the instance will fire **ShapeParentChanged**, **BeforeSelectionDelete**, and **BeforeShapeDelete**, and then ungroup the shapes.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QueryCancelWindowClose event

Example

Occurs before the application closes a window in response to a user action in the interface. If any event handler returns **True**, the operation is canceled.

Version added

2000

Syntax

Private Function *object*_**QueryCancelWindowClose(By**

object	The **WithEvents** object that receives the event.
return value	**False** to allow the operation; **True** to cancel the oper

Remarks

A Visio instance fires **QueryCancelWindowClose** after the user has directed the instance to close a window.

If any event handler returns **True** (cancel), the instance will fire **WindowCloseCanceled** and not close the window.

If all handlers return **False** (don't cancel), the instance will fire **BeforeWindowClosed** then close the window.

While a Visio instance is firing a query or cancel event it will respond to inquiries from client code but it will refuse to perform operations. Client code can show forms or message boxes while responding to a query or cancel event.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

QuitCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelQuit** event.

Version added

2000

Syntax

```
Private Sub object_QuitCanceled(ByVal app As IVAppl
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives

notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

RunModeEntered event

Example

Occurs after a document enters run mode.

Version added

5.0

Syntax

```
Private Sub object_RunModeEntered(ByVal doc As IV
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

SelectionAdded event

Example

Occurs after one or more shapes are added to a document.

Version added

4.5

Syntax

```
Private Sub object_SelectionAdded(ByVal Selection As
```

object The **WithEvents** object that receives the event.

Remarks

A **Shape** object can serve as the source object for the **SelectionAdded** event if the shape's **Type** property is **visTypeGroup**(2) or **visTypePage**(1).

The **SelectionAdded** and **ShapeAdded** events are similar in that they both fire after shape(s) are created. They differ in how they behave when a single operation adds several shapes. Suppose a **Paste** operation creates three new shapes. The **ShapeAdded** event fires three times and acts on each of the three

objects. The **SelectionAdded** event fires once and it acts on a **Selection** object in which the three new shapes are selected.

If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

Note The **SelectionAdded** event is included in the event set of all the objects in the **Applies to** list. For those objects you can use VBA **Dim WithEvents** variables to sink the **SelectionAdded** event.

For performance considerations, the **Document** object's event set does not include the **SelectionAdded** event. To sink the **SelectionAdded** event from a **Document** object (and the **ThisDocument** object in a VBA project), you must use the **AddAdvise** method.

SelectionChanged event

Example

Occurs after a set of shapes selected in a window changes.

Version added

4.5

Syntax

```
Private Sub object_SelectionChanged(ByVal Window A
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

SelectionDeleteCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelSelectionDelete** event.

Version added

2000

Syntax

Private Sub *object*_**SelectionDeleteCanceled**(**ByVal** Sel

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives

notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

ShapeAdded event

Occurs after one or more shapes are added to a document.

Version added

4.1

Syntax

```
Private Sub object_ShapeAdded(ByVal Shape As IVSha
```

object The **WithEvents** object that receives the event.

Remarks

A **Shape** object can serve as the source object for the **ShapeAdded** event if the shape's **Type** property is **visTypeGroup**(2) or **visTypePage**(1).

The **SelectionAdded** and **ShapeAdded** events are similar in that they both fire after shape(s) are created. They differ in how they behave when a single operation adds several shapes. Suppose a **Paste** operation creates three new shapes. The **ShapeAdded** event fires three times and acts on each of the three

objects. The **SelectionAdded** event fires once and it acts on a **Selection** object in which the three new shapes are selected.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

ShapeChanged event

Example

Occurs after a property of a shape that is not stored in a cell is changed in a document.

Version added

4.5

Syntax

```
Private Sub object_ShapeChanged(ByVal Shape As IVS
```

object The **WithEvents** object that receives the event.

Remarks

To determine which properties have changed when **ShapeChanged** fires, use the **EventInfo** property. The string returned by the **EventInfo** property contains a list of substrings that identify the properties that changed.

Changes to the following shape properties cause the **ShapeChanged** event to fire:

Shape name (the **EventInfo** property contains "/name")

Data1 (the **EventInfo** property contains "/data1")

Data2 (the **EventInfo** property contains "/data2")

Data3 (the **EventInfo** property contains "/data3")

UniqueID (the **EventInfo** property contains "/uniqueid")

If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

If you are handling this event from a program that receives a notification over a connection that was created using AddAdvise, the *vMoreInfo* argument to **VisEventProc** contains the string: "/doc=1 /page=1 /shape=Sheet.3"

Note The **ShapeChanged** event is included in the event set of all the objects in the **Applies to** list. For those objects you can use VBA **Dim WithEvents** variables to sink the **ShapeChanged** event.

For performance considerations, the **Document** object's event set does not include the **ShapeChanged** event. To sink the **ShapeChanged** event from a **Document** (and the **ThisDocument** object in a VBA project), you must use the **AddAdvise** method.

ShapeExitedTextEdit event

Example

Occurs after a shape is no longer open for interactive text editing.

Version added

2000

Syntax

```
Private Sub object_ShapeExitedTextEdit(ByVal Shape
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

ShapeParentChanged event

Example

Occurs after shapes are grouped or a group is ungrouped.

Version added

2000

Syntax

```
Private Sub object_ShapeParentChanged(ByVal Shape
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

ShapesDeleted event

Applies to

Occurs after one or more shapes are deleted from a document.

Version added

4.1

Syntax

You can only handle the **ShapesDeleted** event by creating an **Event** object using the **Add** or **AddAdvise** method. See those topics for details about the correct syntax.

Remarks

For performance considerations, the **Document** object's event set does not include the **ShapesDeleted** event. To sink the **ShapesDeleted** event from a **Document** (and the **ThisDocument** object in a VBA project), you must use the **AddAdvise** method.

Because the **ShapesDeleted** event is an after event, the deleted shapes are gone when the notification is received. To receive notification just before shapes are deleted, use the **BeforeShapeDelete**, **BeforeSelectionDelete**, or **BeforeWindowSelDelete** event instead.

How you determine which page or master contained the deleted shapes depends on the **Action** property of the **Event** object whose target has been triggered.

If the event's **Action** property value is **visActCodeRunAddon**, then the index of the document and page, or document and master containing the shapes is passed in the command string.

If the **Action** property value is **visActCodeAdvise**, then the *pSubjectObj* argument passed to **visEventProc** is a **Selection** object whose **ContainingShape** property is the parent shape of the shapes that got deleted, and the *vMoreInfo* argument to **VisEventProc** designates the page or master that contained the deleted shapes.

The **EventInfo** property of the **Application** object returns a string that contains additional information about the names of the deleted shapes:

If one shape is deleted, the string has the following form:

/shapes=shapename

where shapename is the shape's unique ID if it has one; otherwise it is the shape's name ID (sheet.n).

If more than one shape is deleted, the string is in the following form, unless the total number of characters in the string exceeds 8,096 characters:

/shapes=shapename1;shapename2;shapename3;...

If a group is deleted, only the group is included in the string. The group's members are not included.

If the total number of characters in the string exceeds 8,096 characters, it has the following form:

/shapes=many

StyleAdded event

Example

Occurs after a new style is added to a document.

Version added

4.1

Syntax

```
Private Sub object_StyleAdded(ByVal Style As IVStyle
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

StyleChanged event

Example

Occurs after the name of a style is changed or a change to the style propagates to objects to which the style is applied.

Version added

4.1

Syntax

```
Private Sub object_StyleChanged(ByVal Style As IVSty
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives

notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

StyleDeleteCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelStyleDelete** event.

Version added

2000

Syntax

```
Private Sub object_StyleDeleteCanceled(ByVal Style A
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives

notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

SuspendCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelSuspend** event.

Version added

2000 SR-1

Syntax

```
Private Sub object_SuspendCanceled(ByVal app As IVa
```

object	The **WithEvents** object that receives the event.

Remarks

If your solution runs outside of the Visio process you cannot be assured of receiving this event. For this reason, you should monitor window messages in your program.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

TextChanged event

Example

Occurs after the text of a shape is changed in a document.

Version added

4.1

Syntax

```
Private Sub object_TextChanged(ByVal Shape As IVSh
```

object The **WithEvents** object that receives the event.

Remarks

The **TextChanged** event is fired when the raw text of a shape changes, such as when the characters Visio stores for the shape change. If a shape's characters change because a user is typing, the **TextChanged** event does not fire until the text editing session terminates.

When a field is added to or removed from a shape's text, its raw text changes; hence, a **TextChanged** event fires. However, no **TextChanged** event fires when

the text in a field changes. For example, a shape has a text field showing its width. A **TextChanged** event does not fire when the shape's width changes because the raw text stored for the shape hasn't changed, even though the apparent (expanded) text of the shape does change. Use the **CellChanged** event for one of the cells in the Text Fields section to detect when the text in a text field changes.

To access a shape's raw text, use the **Text** property. To access the text of a shape in which text fields have been expanded, use the **Characters.Text** property. You can determine the location and properties of text fields in a shape's text using the **Shape.Characters** object.

In Visio 5.0 and earlier versions, the raw characters reported by the **Text** property for a field included four characters, the first being the Escape character. Starting with Visio 2000, only a single Escape character is present in the raw text stream.

If you're using Microsoft Visual Basic or Visual Basic for Applications (VBA), the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

Note The **TextChanged** event is included in the event set of all the objects in the **Applies to** list. For those objects you can use VBA **Dim WithEvents** variables to sink the **TextChanged** event.

For performance considerations, the **Document** object's event set does not include the **TextChanged** event. To sink the **TextChanged** event from a **Document** object (and the **ThisDocument** object in a VBA project), you must use the **AddAdvise** method.

UngroupCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelUngroup** event.

Version added

2000

Syntax

```
Private Sub object_UngroupCanceled(ByVal Selection
```

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives

notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

ViewChanged event

Example

Occurs when the zoom level or scroll position of a drawing window changes.

Version added

2000

Syntax

```
Private Sub object_ViewChanged(ByVal Window As IV
```

object The **WithEvents** object that receives the event.

Remarks

This event fires whenever the zoom level or scroll position of a **Window** object of the type **visDrawing** changes.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise**

method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

VisioIsIdle event

Example

Occurs after the application empties its message queue.

Version added

5.0

Syntax

```
Private Sub object_VisioIsIdle(ByVal app As IVApplica
```

object The **WithEvents** object that receives the event.

Remarks

Visio continually processes messages in its message queue. When its message queue is empty:

Visio performs its own idle time processing.

Visio tells Microsoft Visual Basic for Applications to perform its idle time processing.

If the message queue is still empty, Visio fires the **VisioIsIdle** event.

If the message queue is still empty, Visio calls **WaitMessage**, which is a call to Microsoft Windows that doesn't return until a new message gets added to the Visio message queue.

A client program can use the **VisioIsIdle** event as a signal to perform its own background processing.

The **VisioIsIdle** event is not the equivalent of a standard timer event. Client programs that need to be called on a periodic basis should use standard timer techniques because the duration in which Visio is idle (calls **WaitMessage**) is unpredictable. For client programs that are only monitoring Visio activity, however, the **VisioIsIdle** event can be sufficient because until **WaitMessage** returns to Visio, there cannot have been any Visio activity since the **VisioIsIdle** event was last fired.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

WindowActivated event

Example

Occurs after the active window changes in a Microsoft Visio instance.

Version added

4.1

Syntax

```
Private Sub object_WindowActivated(ByVal Window A
```

object The **WithEvents** object that receives the event.

Remarks

The **WindowActivated** event indicates that the active window has changed in a Visio instance. This event implies that the **ActiveDocument** and **ActivePage** properties of the **Application** object may also have changed; in contrast, any time the **ActiveDocument** or **ActivePage** property changes, a **WindowActivated** event is always generated.

If you're using Microsoft Visual Basic or Visual Basic for Applications, the

syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

WindowChanged event

Example

Occurs when the size or position of a window changes.

Version added

2000

Syntax

```
Private Sub object_WindowChanged(ByVal Window A
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

WindowCloseCanceled event

Example

Occurs after an event handler has returned **True** (cancel) to a **QueryCancelWindowClose** event.

Version added

2000

Syntax

Private Sub *object*_**WindowCloseCanceled(ByVal** Win

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives

notification, use the **AddAdvise** method. To find an event code for the event you want to create, see Event codes.

WindowOpened event

Example

Occurs after a window is opened.

Version added

4.1

Syntax

```
Private Sub object_WindowOpened(ByVal Window As
```

object The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

WindowTurnedToPage event

Example

Occurs after a window shows a different page.

Version added

4.5

Syntax

```
Private Sub object_WindowTurnedToPage(ByVal Wind
```

object	The **WithEvents** object that receives the event.

Remarks

If you're using Microsoft Visual Basic or Visual Basic for Applications, the syntax in this topic describes a common, efficient way to handle events.

If you want to create your own **Event** objects, use the **Add** or **AddAdvise** method. To create an **Event** object that runs an add-on, use the **Add** method as it applies to the **EventList** collection. To create an **Event** object that receives notification, use the **AddAdvise** method. To find an event code for the event you

want to create, see Event codes.

Activate method

Activates a window.

Version added

2.0

Syntax

object.**Activate**

object	Required. An expression that returns a **Window** object.

Remarks

Visio can have more than one window open at a time; however, only one window is active. Activating a window can change the objects returned by the **ActiveWindow**, **ActivePage**, and **ActiveDocument** properties.

Add method

Adds a new object to a collection.

Version added

2.0

Syntax

objRet = *object*.**Add**

objRet	The new object added to the collection.
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

All properties of the new object are initialized to zero, so you need to set only the properties that you want to change.

Add method (Addons collection)

See also

Adds a new **Addon** object to an **Addons** collection.

Version added

2.0

Syntax

addonObj = *object*.**Add** (*fileName*)

addonObj	The new **Addon** object added to the **Addons** collection.
object	Required. An expression that returns an **Addons** collection.
fileName	Required **String**. The name of the add-on.

Remarks

The **Add** method adds an EXE or VSL file to the collection and returns an **Addon** object if the string expression specifies an EXE file, or **Nothing** if the string expression specifies a VSL file.

Add method (Documents collection)

Adds a new **Document** object to the **Documents** collection.

Version added

2.0

Syntax

docObj = *object*.**Add** (*fileName*)

docObj	The new **Document** object added to the **Documents** collection.
object	Required. An expression that returns a **Documents** collection.
fileName	Required **String**. The type or file name of object to add; if you don't include a path, Visio searches folders designated in the **Application** object's **TemplatePaths** property.

Remarks

To create a new drawing based on no template, pass a zero-length string ("") to the **Add** method.

To create a new drawing based on a template, pass "templatename.vst" to the **Add** method. Visio opens stencils that are part of the template's workspace and copies styles and other settings associated with the template to the new document. If the template file name is invalid, no document is returned and an error is generated.

To create a new stencil based on no stencil, pass ("vss").

To open a copy of a stencil, pass ("stencilname.vss").

To open a copy of a drawing, pass ("drawingname.vsd").

Note Opening a copy of a stencil or drawing is equivalent to selecting **Copy** in the **Open** list box of the **Open** dialog box or using the **OpenEx** method with the **visOpenCopy** flag.

Add method (EventList collection)

Example

Adds an **Event** object that runs an add-on when an event occurs. The **Event** object is added to the **EventList** collection of the source object whose events you want to receive.

Version added

4.1

Syntax

eventObj = object.**Add** (*eventCode*, **visActCodeRunAdd**

eventObj	The new **Event** object added to the **EventList** collection.
object	Required. An expression that returns an **EventList** collect
eventCode	Required **Integer**. The event(s) to capture.
target	Required **String**. The name of your add-on.
targetArgs	Required **String**. The string that is passed to your add-on.

Remarks

The source object whose **EventList** collection contains the **Event** object establishes the scope in which the events are reported. Events are reported for the source object and objects lower in the object model hierarchy. For example, to run an add-on when a particular document is opened, add an **Event** object for the **DocumentOpened** event to the **EventList** collection of that document. To run an add-on when any document is opened in an instance of the application, add the **Event** object to the **EventList** collection of the **Application** object.

Creating **Event** objects is a common way to handle events from C++ or other non–Microsoft Visual Basic solutions. Unlike events handled using the Visual Basic **WithEvents** keyword (all the events in a source object's event set fire), your program will only be notified of the events you select. Depending on your solution, this may result in improved performance.

Event objects that run add-ons can be persistent, that is, they can be stored with a Visio document. To be persistent, an **Event** object's **Persistent** and **Persistable** properties must both be **True**.

The arguments passed to the **Add** method set the initial values of the **Event** object's **Event**, **Action (visCodeRunAddon)**, **Target**, and **TargetArgs** properties.

Event codes are declared by the Visio type library and have the prefix **visEvt**. Event codes are often a combination of constants. For example, **visEvtAdd+visEvtDoc** is the event code for the **DocumentAdded** event. To find an event code for the event you want to create, see Event codes.

To create an **Event** object that advises the caller's sink object about an event, see the **AddAdvise** method.

Add method (Layer object)

Assigns a shape to a layer.

Version added

2.0

Syntax

object.**Add**(*shapeObj*, *fPreserveMembers*)

object	Required. An expression that returns a **Layer** object.
shapeObj	Required. The new **Shape** object added to the **Layer** object.
fPreserveMembers	Required **Integer**. Zero to remove subshapes from any previous layer assignments; non-zero to preserve layer assignments.

Remarks

If the shape is a group and *fPreserveMembers* is non-zero, the component shapes of the group retain their current layer assignments and are also added to this layer. If *fPreserveMembers* is zero, the component shapes are reassigned to this

layer and lose their current layer assignments.

Add method (Layers collection)

See also

Adds a new **Layer** object to a **Layers** collection.

Version added

2.0

Syntax

layerObj = *object*.**Add** (*layerName*)

layerObj	The new **Layer** object added to the **Layers** collection.
object	Required. An expression that returns a **Layers** collection.
layerName	Required **String**. The name of the new layer.

Add method (Styles collection)

Adds a new **Style** object to a **Styles** collection.

Version added

2.0

Syntax

styleObj = object.**Add**(*newStyleName*, *basedOnName*, *fIn*

styleObj	The new **Style** object added to the **Styles** collection.
object	Required. An expression that returns a **Styles** collecti
newStyleName	Required **String**. The new style name.
basedOnName	Required **String**. The name of the style on which to b
fIncludesText	Required **Integer**. Zero to disable text attributes, or n
fIncludesLine	Required **Integer**. Zero to disable line attributes, or n
fIncludesFill	Required **Integer**. Zero to disable fill attributes, or nc

Remarks

Pass a zero-length string ("") for the *basedOnName* argument to base the new style on no style.

Add method (Windows collection)

Example

Adds a new **Window** object to the **Windows** collection.

Version added

2000

Syntax

objRet = *object*.**Add** ([*varCaption*][, *varFlags*]
[, *varType*][, *varLeft*][, *varTop*][, *varWidth*]
[, *varHeight*][, *bstrMergeID*][, *bstrMergeClass*]
[, *nMergePostition*])

objRet	The new **Window** object added to the collection.
object	Required. An expression that returns a **Windows** collection.
varCaption	Optional **Variant**. The title of window; default is "Untitled".
varFlags	Optional **Variant**. Initial window state. Can contain any combination of **visWindowStates** constants declared in the Visio type library; default varies based on the *varType*.

varType	Optional **Variant**. Type of new window. Defaults to **visStencilAddon** for **Application.Windows**; defaults to **visAnchorBarAddon** for **Window.Windows**.
varLeft	Optional **Variant**. Position of the left side of the window.
varTop	Optional **Variant**. Position of the top of the window.
varWidth	Optional **Variant**. Width of the client area of the window.
varHeight	Optional **Variant**. Height of the client area of the window.
bstrMergeID	Optional **Variant**. Merge ID of the window.
bstrMergeClass	Optional **Variant**. Merge class of the window.
nMergePostition	Optional **Variant**. Merge position of the window.

Remarks

Use this method to get an empty parent frame window within the Visio window space that you can populate with child windows. You must be in the Visio process space (for example, in a DLL/VSL based add-on) to use the **Window** object returned by this method as a parent to your windows.

Use the value returned by the **WindowHandle32** property as an **HWND** for use as a parent to your own windows.

AddAdvise method

Adds an **Event** object to the **EventList** collection of the source object whose events you want to receive. When selected events occur, the source object will notify your sink object.

Version added

4.1

Syntax

evtObj = object.**AddAdvise** (*eventCode*, *eventSink*, *IIDSi*

evtObj	The new **Event** object.
object	Required. An expression that returns an **EventList** collectio
eventCode	Required **Integer**. The event(s) that generate notifications.
eventSink	Required **Object**. A reference to an OLE interface on the ob receive event notifications.
IIDSink	Required **String**. Reserved for future use. Must be "".
targetArgs	Required **String**. The string that is passed to your sink objec

Remarks

Event objects created with the **AddAdvise** method have an **Action** property of **visActCodeAdvise**. They are not persistent, that is, they cannot be stored with a Visio document and must be re-created at run time.

The source object whose **EventList** collection contains the **Event** object establishes the scope in which the events are reported. Events are reported for the source object and objects lower in the object model hierarchy. For example, to receive notification when a particular document is saved, add an **Event** object for the **DocumentSaved** event to the **EventList** collection of that document. To receive notification when any document is opened in an instance of the application, add the **Event** object to the **EventList** collection of the **Application** object.

Creating **Event** objects is a common way to handle events from C++ or other non–Microsoft Visual Basic solutions. Unlike events handled using the Visual Basic **WithEvents** keyword (all the events in a source object's event set fire), your program will only be notified of the events you select. Depending on your solution, this may result in improved performance.

The *eventCode* argument is often a combination of constants. For example, **visEvtMod+visEvtCell** is the event code for the **CellChanged** event. Event constants are declared by the Visio type library and are prefixed with **visEvt**. To find an event code for the event you want to create, see Event codes.

The arguments passed to the **AddAdvise** method set the initial values of the **Event** object's **Event**, **Action** (**visCodeRunAddAdvise**), and **TargetArgs** properties.

Beginning with Microsoft Visio 2002, you can use event filters to refine the events that you receive in your program. You can filter events by object, cell, ranges of cells, or command ID. For details about using event filters, see the method topics prefixed with **SetFilter** and **GetFilter**.

- Enabling your program to handle event notifications from Microsoft Visual Basic or Visual Basic for Applications

AddAt method

Creates a new object at a specified index in a collection.

Version added

4.0

Syntax

objRet = *object*.**AddAt**(*index*)

objRet	The new object added to the collection.
object	Required. An expression that returns an object in the **Applies to** list.
index	Required **Long**. The index at which to add the object.

Remarks

If the index is zero (0), the object is added at the beginning of the collection.

The beginning of a **Menus** collection is the leftmost menu when the menus are arranged horizontally. For example, the **File** menu is the first menu in the **Menus**

collection for the drawing window context.

The beginning of a **MenuItems** collection is the topmost menu item. For example, the **New Window** menu item is the first menu item in the **MenuItems** collection for the Window **Menu** object.

The beginning of a **ToolbarItems** collection is the leftmost item in a toolbar that is arranged horizontally.

AddAtID method

Example

Creates a new object with a specified ID in a collection.

Version added

4.0

Syntax

objRet = *object*.**AddAtID**(*id*)

objRet	The new object added to the collection.
object	Required. An expression that returns an **AccelTables**, **MenuSets**, or **ToolbarSets** collection.
id	Required **Long**. The window context for the new object.

Remarks

The ID corresponds to a window or context menu. If the collection already contains an object at the specified ID, the **AddAtID** method returns an error.

Valid IDs are declared by the Visio type library and begin with **visUIObjSet**.

Not all collections include an object for every possible ID. For a list of valid contexts for a particular collection, see the **SetID** property.

AddCustomField[U] method

Example

Replaces the text represented by a **Characters** object with a custom formula field.

Version added

3.0

Syntax

object.**AddCustomField** *strFormula*, *intFormat*

object	Required. An expression that returns a **Characters** object.
strFormula	Required **String**. The formula of the new field.
intFormat	Required **Integer**. The format of the new field.

Remarks

Using the **AddCustomField** method is similar to clicking **Field** on the **Insert** menu and inserting a custom formula field in text. To add any other type of field (not custom), use the **AddField** method.

For a list of valid field format constants, see the **FieldFormat** property. Valid field format constants are also defined in the Visio type library in **VisFieldFormats**.

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **AddCustomField** method to set a custom field using local syntax. Use the **AddCustomFieldU** method to set a custom field using universal syntax.

AddField method

Example

Replaces the text represented by a **Characters** object with a new field of the category, code, and format you specify.

Version added

3.0

Syntax

object.**AddField** *intCategory*, *intCode*, *intFormat*

object	Required. An expression that returns a **Characters** object.
intCategory	Required **Integer**. The category for the new field.
intCode	Required **Integer**. The code for the new field.
intFormat	Required **Integer**. The format for the new field.

Remarks

Using the **AddField** method is similar to clicking **Field** on the **Insert** menu, and inserting any of the following categories of fields in the text:

Date/Time

Document Info

Geometry

Object Info

Page Info

To add a custom formula field, use the **AddCustomField** method.

For *intCategory, intCode, and intFormat* constant values, see the **FieldCategory**, **FieldCode**, and **FieldFormat** property topics. These constants are also declared by the Visio type library in **VisFieldCategories**, **VisFieldCodes**, and **VisFieldFormats**.

AddGuide method

Adds a guide to a drawing page.

Version added

2.0

Syntax

objRet = *object*.**AddGuide** (*guideType*, *x*, *y*)

objRet	A **Shape** object that represents the new guide.
object	Required. An expression that returns a **Page** object.
guideType	Required **Integer**. The type of guide to add.
x	Required **Double**. The *x*-coordinate of a point on the guide.
y	Required **Double**. The *y*-coordinate of a point on the guide.

Remarks

The following constants declared by the Visio type library are valid values for guides.

Constant	Value	Description
visPoint	1	Guide point
visHorz	2	Horizontal guide
visVert	3	Vertical guide

AddHyperlink method

Adds a **Hyperlink** object to a Microsoft Visio shape.

Version added

5.0

Syntax

objRet = object.**AddHyperlink**

objRet	The **Hyperlink** object that is returned.
object	Required. An expression that returns a **Shape** object.

Remarks

Using the **AddHyperlink** method is equivalent to adding a hyperlink to a shape by clicking **Hyperlinks** on the **Insert** menu.

If a **Hyperlink** object already exists for the shape, then a reference to the existing **Hyperlink** object is returned.

AddNamedRow method

Example

Adds a row with the specified name to the specified ShapeSheet section.

Version added

4.0

Syntax

retVal = object.**AddNamedRow** (*section*, *rowName*, *rowT*

retVal	**Integer**. The row number of the new row.
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The section in which the row is to be added.
rowName	Required **String**. The name of the new row.
rowTag	Required **Integer**. The type of row to be added.

Remarks

You can add named rows to the Custom Properties (**visSectionProp**), User-defined Cells (**visSectionUser**), and Connection Points

(**visSectionConnectionPts**) ShapeSheet sections. You can access cells in the new rows by passing the row number returned by the **AddNamedRow** method to the **CellsSRC** property. Alternatively, you can access cells in the new rows using the row's name with the **Cells** property. For details about cell references and cells in named rows, see the User.Row, Prop.Name, or Connections.Row row topics.

An empty row name string ("") creates a row with a default name.

A value of zero (0) in the *rowTag* argument generates the default row type for the section. Explicit tags are useful when adding rows to the Connection Points section. See the **RowType** property for descriptions of valid row types for each section. Passing an invalid row type generates an error.

Adding a named row to a Connection Points section automatically converts any existing unnamed rows in the section into named rows, using their default names (Row_1, Row_2, and so on).

AddRow method

Adds a row to a ShapeSheet section at a specified position.

Version added

2.0

Syntax

retVal = *object*.**AddRow** (*section*, *row*, *tag*)

retVal	**Integer**. The row number of the row that was added.
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The section in which to add the row.
row	Required **Integer**. The position at which to add the row.
tag	Required **Integer**. The type of row to add.

Remarks

If the ShapeSheet section does not already exist, it is created with a blank row. New cells in new rows are initialized with default formulas, if applicable.

Otherwise, a program must include statements to set the formulas for the new cells. An error is generated if the new row cannot be added.

The row constants declared by the Visio type library serve as base positions at which a section's rows begin. Add offsets to these constants to specify the first row and beyond, for example, **visRowFirst**+0, **visRowFirst**+1, and so on. To add rows at the end of a section, pass the constant **visRowLast** for the row argument. The value returned is the actual row index.

The tag argument specifies the type of row to add. Pass zero (0) as the tag argument to generate a section's default row type. Explicit tags are useful when adding rows to Geometry, Connection Points, and Controls sections. See the **RowType** property for descriptions of valid row types for these sections. Passing an invalid row type generates an error.

If you try to add a row to a Character, Tabs, or Paragraph section, an error occurs.

The **AddRow** method cannot add named rows; an error occurs if the section contains named rows or can hold only named rows. To add named rows, use the **AddNamedRow** method.

The Visio type library declares row constants prefixed with **visRow** in **VisRowIndices**. These are also listed in the **AddRows** method topic.

Constants for rows in the Geometry, Connection Points and Controls sections are prefixed with **visTag** and declared by the type library in **VisRowTags**. To see a list of these constants, see the **RowType** property.

AddRows method

Example

Adds the specified number of rows to a ShapeSheet section at a specified position.

Version added

4.0

Syntax

retVal = *object*.**AddRows** (*section*, *row*, *tag*, *count*)

retVal	**Integer**. The row number of the last row that was added.
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The section in which to add the rows.
row	Required **Integer**. The position at which to add the rows.
tag	Required **Integer**. The type of rows to add.
count	Required **Integer**. The number of rows to add.

Remarks

If the ShapeSheet section does not exist, the **AddRows** method creates a section with blank rows. New cells in new rows are initialized with default formulas, if applicable. Otherwise, a program must include statements to set the formulas for the new cells. An error occurs if the row cannot be added.

The row constants declared by the Visio type library serve as base positions at which a section's rows begin. Add offsets to these constants to specify the first row and beyond, for example, **visRowFirst**+0, **visRowFirst**+1, and so on. To add rows at the end of a section, pass the constant **visRowLast** for the row argument. The value returned is the actual row index.

The tag argument specifies the type of rows to add. Pass zero (0) as the tag argument to generate a section's default row type. Explicit tags are useful when adding rows to Geometry, Connection Points, and Controls sections. See the **RowType** property for descriptions of valid row types for these sections. Passing an invalid row type generates an error.

If you try to add rows to a Character, Tabs, or Paragraph section, an error occurs.

The **AddRows** method cannot add named rows; an error occurs if the section contains named rows or can hold only named rows. To add named rows, use the **AddNamedRow** method.

The Visio type library declares the constants for *tag* in **VisRowIndices**.

- Valid constants for *tag*

AddSection method

Adds a new section to a ShapeSheet spreadsheet.

Version added

2.0

Syntax

intRet = *object*.**AddSection** (*section*)

intRet	**Integer**. The index of the section that was added.
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The type of section to add.

Remarks

The **AddSection** method is frequently used to add one or more Geometry sections to a shape. You can also use **AddSection** to add other sections to a shape such as Scratch, Controls, Connection Points, Actions, User-Defined Cells and Custom Properties. The **AddSection** method returns the logical index of the added section.

The sections that you can add to a shape correspond to the choices shown by the **Insert Section** dialog box when the shape is displayed in a ShapeSheet window.

If you try to add a non-Geometry section to a shape that already has that section, the **AddSection** method raises an exception. Use the **SectionExists** property to determine if a shape already has a section with a given logical index.

A new section has no rows. Use the **AddRow** method to add rows to the new section.

The **GeometryCount** property returns the number of Geometry sections included in a shape. Use the following code to add a Geometry section to a shape:

```
Shape.AddSection(visSectionFirstComponent+i)
```

where 0 <= *i* < **visSectionLastComponent - visSectionFirstComponent**. The new section precedes the present *i*'th Geometry section for 0 <= *i* < **Shape.GeometryCount**. It is the last section for **Shape.GeometryCount** <= *i* < **visSectionLastComponent - visSectionFirstComponent**.

The Visio type library declares the constants for sections in **VisSectionIndices**.

- Valid section constants

AddToFavorites method

Adds a shortcut for a hyperlink address in the presently registered Favorites folder.

Version added

5.0

Syntax

object.**AddToFavorites** [*favoritesTitle*]

object	Required. An expression that returns a **Hyperlink** object.
favoritesTitle	Optional **String**. The title to assign to the new shortcut.

Remarks

If a string is not supplied, the **AddToFavorites** method uses the hyperlink's **Description** property as the new favorite's title. If the **Description** property is empty, the shortcut is given a generic title, such as Favorite1.

The optional *favoritesTitle* argument can specify the full path for the favorites

file, for example, "C:\TEMP\My Favorite.URL", or a path relative to the favorites folder.

From Microsoft Visual Basic or Visual Basic for Applications, a call to the **AddToFavorites** method can take either of these two forms:

```
object.AddToFavorites "SomeString"
object.AddToFavorites
```

From C/C++, if a string is supplied, pass a **Variant** of type VT_BSTR. The application assigns the string as the title of the shortcut. If a string is not supplied, pass a **Variant** of type VT_EMPTY, or of type VT_ERROR and HRESULT DISP_E_PARAMNOTFOUND.

AddToGroup method

Adds the selected shapes to the selected group.

Version added

2.0

Syntax

object.**AddToGroup**

object	Required. An expression that returns a **Selection** object.

Remarks

The current selection must contain both the shapes to add and the group to which you want to add them. The group must be the primary selection or the only group in the selection.

AddUndoUnit method

Adds an object that supports the **IOleUndoUnit** or **IVBUndoUnit** interface to the Microsoft Visio undo queue.

Version added

2000

Syntax

object.**AddUndoUnit** (*object*)

object	Required. An expression that returns an **Application** object.
object	Required. An object that supports the **IOleUndoUnit** or **IVBUndoUnit** interface.

Remarks

For information about implementing the **IOleUndoUnit** interface on your object, see the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site. For information about implementing the **IVBUndoUnit** interface, see Developing Visio Solutions on the MSDN Web site.

Arrange method

Arranges the windows in a **Windows** collection.

Version added

2.0

Syntax

object.**Arrange**(*nArrangeFlags*)

object	Required. An expression that returns a **Windows** collection.
nArrangeFlags	Optional **Variant**. A flag that specifies how to arrange the windows; by default, the windows are arranged vertically.

Remarks

Using the **Arrange** method is equivalent to clicking **Tile** on the **Window** menu. The active window remains active.

Visio considers windows top to bottom, then left to right. You can influence which windows will end up topmost when tiling horizontally (or leftmost when

tiling vertically) by prearranging windows.

The following constants declared by the Visio type library are valid values for *nArrangeFlags*. These constants are also declared by the Visio type library in **VisWindowArrange**.

Constant	Value
VisArrangeTileVertical	1
VisArrangeTileHorizontal	2
VisArrangeCascade	3

Example

The following macro uses the **SetWindowRect** method to prearrange windows. This creates the desired results when the **Arrange** method is called to tile the windows. In this example, window(*i*) ends up above window(*i*+1).

```
Public Sub p()
    For i = 1 To Windows.Count
        Windows(i).SetWindowRect i + 100, i + 100, 100, 1
    Next i
    Windows.Arrange visArrangeTileHorizontal
End Sub
```

BeginUndoScope method

Starts a transaction with a unique scope ID for an instance of Microsoft Visio.

Version added

2000

Syntax

nScopeID = *object*.**BeginUndoScope** (*stringDescription*)

nScopeID	**Long**. The ID of the new scope within the Visio instance.
object	Required. An expression that returns an **Application** object.
stringDescription	Required **String**. The name of the scope; could appear in the Visio user interface.

Remarks

If you need to know whether events you receive are the result of a particular operation that you initiated, use the **BeginUndoScope** and **EndUndoScope** methods to wrap your operation. In your event handlers, use the **IsInScope** property to test whether the scope ID returned by the **BeginUndoScope** method

is part of the current context. Make sure you clear the scope ID you stored from the **BeginUndoScope** property when you receive the **ExitScope** event with that ID.

You must balance calls to the **BeginUndoScope** method with calls to the **EndUndoScope** method. If you call the **BeginUndoScope** method, you should call the **EndUndoScope** method as soon as you are finished with the actions that constitute your scope. Also, while actions to multiple documents should be robust within a single scope, closing a document may have the side effect of purging the undo information for the currently open scope as well as purging the undo and redo stacks. If that happens, passing *bCommit* = **False** to **EndUndoScope** does not restore the undo information.

You can also use the **BeginUndoScope** and **EndUndoScope** methods to add an action defined by an add-on to the Visio undo stream. This is useful when you are operating from modeless scenarios where the initiating agent is part of an add-on's user interface or a modeless programmatic action.

Note Most Visio actions are already wrapped in internal undo scopes, so add-ons running within the application do not need to call this method.

BoundingBox method

Returns a rectangle that tightly encloses a shape, or the shapes of a page, master, or selection.

Version added

4.5

Syntax

object.**BoundingBox** *flags*, *left*, *bottom*, *right*, *top*

object	Required. An expression that returns the **Page**, **Master**, **Shape** or **Selection** object whose bounding box is to be retrieved.
flags	Required **Integer**. Flags that influence the bounding box calculated for each shape that contributes to the resulting bounding box.
left	Required **Double**. Returns *x*-coordinate of left edge of bounding box.
bottom	Required **Double**. Returns *y*-coordinate of bottom edge of bounding box.

right	Required **Double**. Returns *x*-coordinate of right edge of bounding box.
top	Required **Double**. Returns *y*-coordinate of top edge of bounding box.

Remarks

For a **Shape** object, the **BoundingBox** method returns a rectangle that tightly encloses the shape and its sub-shapes.

For a **Page**, **Master**, or **Selection** object, the **BoundingBox** method returns a rectangle that tightly encloses the page's, master's, or selection's shapes and their sub-shapes.

If the **BoundingBox** method returns an error, or if it is asked to return the rectangle enclosing zero shapes, the rectangle returned is { left: 0, bottom: 0, right: -1, top: -1 }; otherwise, the rectangle returned has left less than or equal to (<=) right and bottom less than or equal to (<=) top. The numbers returned are in internal units (inches).

The bounding rectangle returned for an individual shape depends on its **Type** property.

Constant	Description
visTypePage	Equivalent to **Page.BoundingBox** or **Master.BoundingBox**.
visTypeGroup	Rectangle that tightly encloses the group and its sub-shapes.
visTypeShape	Determined rectangle depends on *flags*. See below.
visTypeForeignObject	Determined rectangle depends on *flags*. See below.
visTypeGuide	Determined rectangle depends on *flags*. See below.

The method will raise an exception for object type **visTypeDoc**.

The *flags* argument has several bits that control the bounding box retrieved for

each shape. If more than one of the bits described below is set, the rectangle determined for the shape covers all rectangles implied by the bits.

Flag	Value	Description
visBBoxUprightWH	&H1	Return a rectangle that is the smallest rectangle parallel to the local coordinate system of the shape's parent that encloses the shape's width-height box. If the shape is not rotated, its upright width-height box and its width-height box are the same. Paths in the shape's geometry needn't and often don't lie entirely within the shape's width-height box.
visBBoxUprightText	&H2	Return a rectangle that is the smallest rectangle parallel to the local coordinate system of the shape's parent that encloses the shape's text.
visBBoxExtents	&H4	Return a rectangle that is the smallest rectangle parallel to the local coordinate system of the shape's parent that encloses the paths stroked by the shape's geometry. This may be larger or smaller than the shape's upright width-height box. The extents box determined for a shape of type **visTypeForeignObject** equals that shape's upright width-height box.
visBBoxIncludeHidden	&H10	Include hidden geometry.
visBBoxIgnoreVisible	&H20	Ignore visible geometry.
VisBBoxIncludeGuides	&H1000	Include extents for shapes of type **visTypeguide**. By default, the extents of shapes of type **visTypeGuide** are ignored. If you request guide extents, then only the *x* positions of vertical guides and the *y* positions of horizontal guides contribute to the rectangle that is

		returned. If any vertical guides are reported on, an infinite *y* extent is returned. If any horizontal guides are reported on, an infinite *x* extent is returned. If any rotated guides are reported on, infinite *x* and *y* extents are returned.
visBBoxDrawingCoords	&H2000	Return numbers in the drawing coordinate system of the page or master whose shapes are being considered. By default, the returned numbers are drawing units in the local coordinate system of the parent of the considered shapes.
visBBoxNoNonPrint	&H4000	Ignore the extents of shapes that are non-printing. A shape is non-printing if the value of its NonPrinting cell is non-zero or it belongs only to non-printing layers.

The extents rectangle is determined using the center of the shape's strokes; it does not take into account the width of the strokes. Nor does the rectangle include any area covered by shadows or line end markers. Visio doesn't expose a means to determine a shape's "black bits" box, that is, the extents box adjusted to account for stroke widths, shadows, and line ends.

A shape may have control points or connection points that lie outside any of the bounding rectangles reported by the shape. You can determine the position of control points and connection points by querying results of the shape's cells.

BringForward method

Brings the shape or selected shapes forward one position in the *z*-order.

Version added

2.0

Syntax

object.**BringForward**

object	Required. An expression that returns the **Shape** or **Selection** object to bring forward.

BringToFront method

Brings the shape or selected shapes to the front of the z-order.

Version added

2.0

Syntax

object.**BringToFront**

object	Required. An expression that returns the **Shape** or **Selection** object to bring to the front.

CenterDrawing method

See also Example

Centers a page's, master's, or group's shapes with respect to the extent of the page, master, or group.

Version added

4.0

Syntax

object.CenterDrawing

object	Required. An expression that returns a **Page**, **Master**, or **Shape** object that contains the shapes to center.

Remarks

Centering shapes does not change their position relative to each other.

ClearCustomMenus method

See also

Restores the built-in Microsoft Visio menus.

Version added

4.0

Syntax

object.**ClearCustomMenus**

object	Required. An expression that returns an **Application** or **Document** object that is using the custom menus.

Remarks

Calling the **ClearCustomMenus** method on an object without custom menus has no effect.

ClearCustomToolbars method

See also

Restores the built-in Microsoft Visio toolbars.

Version added

4.0

Syntax

object.**ClearCustomToolbars**

object	Required. An expression that returns an **Application** or **Document** object that is using the custom toolbars.

Remarks

Calling the **ClearCustomToolbars** method on an object without custom toolbars has no effect.

ClearGestureFormatSheet method

Example

Clears local formatting in a document's Gesture Format sheet.

Version added

2000

Syntax

object.**ClearGestureFormatSheet**

object	Required. An expression that returns a **Document** object.

Remarks

Any shapes drawn after the Gesture Format sheet is cleared inherit their line, fill, and text formatting from the document's default styles.

A document's Gesture Format sheet also gets cleared automatically when the document is opened.

For details about the Gesture Format sheet, see the **GestureFormatSheet**

property.

Close method

Closes a window, document, or master.

Version added

2.0

Syntax

object.**Close**

object	Required. An expression that returns the **Window**, **Document**, or **Master** object to close.

Remarks

If the indicated window is the only window open for a document and the document contains unsaved changes, an alert appears asking if you want to save the document. You can use the **AlertResponse** property to prevent the alert from appearing.

If you close a docked stencil window, only that window is closed. However, if

you close a drawing window that contains docked stencils, the docked stencil window is also closed.

Use the **Close** method for a **Master** object after opening a master for editing using the **Open** method. The **Close** method pushes any changes made to the master while it was open to instances of the master.

Combine method

Example

Creates a new shape by combining selected shapes.

Version added

2.0

Syntax

object.**Combine**

object	Required. An expression that returns the **Selection** object that contains the shapes to combine.

Remarks

The **Combine** method is equivalent to clicking the **Combine** command on the **Operations** submenu on the **Shape** menu in Visio. The produced shape will be the topmost shape in its containing shape and will inherit the text and formatting of the first selected shape. The original shapes are deleted and no shapes are selected when the operation is complete.

The **Combine** method is similar to the **Join** method but differs in the following ways:

The **Combine** method produces a shape with one Geometry section for each original shape. The resulting shape will have holes in regions where the original shapes overlapped.

The **Join** method differs from **Combine** in that it will coalesce abutting line and curve segments in the original shapes into a single Geometry section in the resulting shape.

ConvertResult method

Converts a string or number into an equivalent number in different measurement units.

Version added

4.5

Syntax

retVal = object.**ConvertResult**(*StringOrNumber*, *unitsIn*,

retVal	**Double**. The result of the conversion.
object	Required. An expression that returns an **Application** object.
StringOrNumber	Required **Variant**. String or number to be converted; can be a floating point number, or integer.
unitsIn	Required **Variant**. Measurement units to attribute to *StringOr*
unitsOut	Required **Variant**. Measurement units to express the result in.

Remarks

If passed as a string, *StringOrNumber* might be the formula or prospective formula of a cell or the result or prospective result of a cell expressed as a string. The **ConvertResult** method evaluates the string and converts the result into the units designated by *unitsOut*. The **ConvertResult** method returns an error if the string contains any cell references.

Possible values for *StringOrNumber* include:

1.7

3

"2.5"

"4.1 cm"

"12 ft - 17 in + (12 cm / SQRT(7))"

The *unitsIn* and *unitsOut* arguments can be strings such as "inches", "inch", "in.", or "i". Strings may be used for all supported Visio units such as centimeters, meters, miles, and so on. You can also use any of the units constants declared by the Visio type library in **VisUnitCodes**. A list of valid units is also listed in About units of measure.

If *StringOrNumber* is a floating point number or integer, *unitsIn* declares what unit of measure the **ConvertResult** method should construe the number to be. Pass "" to indicate internal Visio units.

If *StringOrNumber* is a string, *unitsIn* specifies how to interpret the evaluated result and is only used if the result is a scalar. For example, the expression "4 * 5 cm" evaluates to 20 cm, which is not a scalar so *unitsIn* is ignored. The expression "4 * 5" evaluates to 20 which is a scalar and is interpreted using the specified *unitsIn*.

The *unitsOut* argument specifies in what units the returned number should be expressed. If you want the results expressed in the same units as the evaluated expression, pass "NOCAST" or **visNoCast**.

Examples where string is specified:

```
Debug.Print application.ConvertResult("0.5 * 2", "ft", "ft"
Debug.Print application.ConvertResult("0.5 * 2", "ft", "in"
Debug.Print application.ConvertResult("1 cm", "ft", "in")
Debug.Print application.ConvertResult("1 cm", "ft", "NO(
Debug.Print application.ConvertResult("1 cm", "ft", "")
Debug.Print application.ConvertResult("1 cm", "ft", "bozo
```

Examples where number is specified:

```
Debug.Print application.ConvertResult(1, "ft", "ft")    >>>
Debug.Print application.ConvertResult(1, "ft", "in")    >>>
Debug.Print application.ConvertResult(1.0, "in", "ft")    >
Debug.Print application.ConvertResult(1.0, visFeet, "")
Debug.Print application.ConvertResult(1, "bozo", "in")    >
```

ConvertToGroup method

Example

Converts a selection or an object from another application (a linked or embedded object) to a group.

Version added

2.0

Syntax

object.**ConvertToGroup**

object	Required. An expression that returns the **Shape** or **Selection** object to convert.

Remarks

If the object to convert is a metafile it will be converted into basic shapes.

Copy method

Copies a text range or hyperlink to the Clipboard.

Version added

2.0

Syntax

object.**Copy**

object	Required. An expression that returns a **Characters** or **Hyperlink** object.

Remarks

To make a copy without using the Clipboard, use the **Duplicate** method.

Copy method (Selection object)

Example

Copies a selection to the Clipboard.

Version added

2002

Syntax

object.**Copy** [*flags*]

object	Required. An expression that returns a **Selection** object.
flags	Optional **Variant**. Determines how shapes are translated during the copy operation.

Remarks

Possible values for *flags* are declared by the Visio type library in **VisCutCopyPasteCodes**, and are described in the following table.

Flag	Value	Description
visCopyPasteNormal	&H0	Default. Shapes are copied to the

		center of the document.
visCopyPasteNoTranslate	&H1	Shapes are copied to their original coordinate locations.

Setting *flag* to **visCopyPasteNormal** is the equivalent of the behavior in the user interface. You should use the **visCopyPasteNormal** and **visCopyPasteNoTranslate** *flags* consistently. For example, if you copy using **visCopyPasteNoTranslate**, you should also paste using that value as it is the only way to ensure that shapes are pasted to their original coordinate location.

To make a copy without using the Clipboard, use the **Duplicate** method.

Copy method (Shape object)

Example

Copies a shape to the Clipboard.

Version added

2002

Syntax

object.**Copy** [*flags*]

object	Required. An expression that returns a **Shape** object.
flags	Optional **Variant**. Determines how shapes are translated during the copy operation.

Remarks

Possible values for *flags* are declared by the Visio type library in **VisCutCopyPasteCodes**, and are described in the following table.

Flag	Value	Description
visCopyPasteNormal	&H0	Default. Shapes are copied to the

		center of the document.
visCopyPasteNoTranslate	&H1	Shapes are copied to their original coordinate locations.

Setting *flag* to **visCopyPasteNormal** is the equivalent of the behavior in the user interface. You should use the **visCopyPasteNormal** and **visCopyPasteNoTranslate** *flags* consistently. For example, if you copy using **visCopyPasteNoTranslate**, you should also paste using that value as it is the only way to ensure that shapes are pasted to their original coordinate location.

To make a copy without using the Clipboard, use the **Duplicate** method.

CopyPreviewPicture method

Example

Copies the preview picture from another document into the current document.

Version added

2002

Syntax

object.**CopyPreviewPicture** *pSourceDoc*

object	Required. An expression that returns a **Document** object.
pSourceDoc	Required. The **Document** object whose preview picture you want to copy into this document.

CreateShortcut method

Example

Creates a shortcut for a master.

Version added

2000

Syntax

objRet = object.**Create**

objRet	The new **MasterShortcut** object.
object	Required. An expression that returns a **Master** object.

Remarks

The new master shortcut is created in the same document as the target master and is added to the document's **MasterShortcuts** collection. The document must therefore be editable for this method to succeed.

The new shortcut's name is "Shortcut to *X*", where "*X*" is the name of the target master. The shortcut's **TargetDocumentName** and **TargetMasterName**

properties identify the target master. So once a shortcut has been created, it can be moved or copied into other documents.

You cannot create a shortcut to a master in an unsaved stencil. If you try to do so, the **CreateShortcut** method returns an error.

CreateURL method

Returns a fully qualified and optionally canonicalized representation of the hyperlink's absolute address.

Version added

5.0

Syntax

strRet = object.**CreateURL**(*intExpression*)

strRet	**String**. A fully qualified URL representation of a hyperlink.
object	Required. An expression that returns a **Hyperlink** object.
intExpression	Required **Integer**. **True** (non-zero) if canonical form; otherwise, **False** (0).

Remarks

The **CreateURL** method of the **Hyperlink** object can be used to resolve relative URLs against a hyperlink's base address.

When you use the canonical form, the **CreateURL** method applies URL canonicalization rules to the hyperlink. Only spaces are URL encoded during canonicalization. Port 80 is assumed for HTTP URLs and is removed during canonicalization. The URL "http://www.microsoft.com:80/" is returned as "http://www.microsoft.com/", whereas http://www.microsoft.com:1000/" is unchanged.

Example

Here are some examples of results of the **CreateURL** method:

```
Address = "http://www.microsoft.com/"
CreateURL(False) returns "http://www.microsoft.com/"

Address = "C:\My Documents\Spreadsheet.XLS"
CreateURL(False) returns "file://C:\My Documents\Sprea
CreateURL(True) returns "file://C:\My%20Documents\Sp
```

Relative path example:

```
Assume : Document.HyperlinkBase = "http://www.micros
Address = "../file.htm"
CreateURL(False) returns "http://www.microsoft.com/file
```

Cut method

Deletes an object or selection and places it on the Clipboard.

Version added

2002

Syntax

object.**Cut** [*flags*]

object	Required. An expression that returns a **Selection** or **Shape** object.
flags	Optional **Variant**. Determines how shapes are translated during the cut operation.

Remarks

Possible values for *flags* are declared by the Visio type library in **VisCutCopyPasteCodes**, and are described in the following table.

Flag	Value	Description

visCopyPasteNormal	&H0	Default. Shapes are copied to the center of the document.
visCopyPasteNoTranslate	&H1	Shapes are copied to their original coordinate locations.

Setting *flags* to **visCopyPasteNormal** is the equivalent of the behavior in the user interface. You should use the **visCopyPasteNormal** and **visCopyPasteNoTranslate** *flags* consistently. For example, if you copy using **visCopyPasteNoTranslate**, you should also paste using that value as it is the only way to ensure that shapes are pasted to their original coordinate location.

Cut method (Characters object)

Example

Deletes a text range and places it on the Clipboard.

Version added

2.0

Syntax

object.**Cut**

object	Required. An expression that returns a **Characters** object.

Remarks

When used with a **Characters** object, the **Cut** method places the text range represented by that object onto the Clipboard.

Delete method

Deletes an object or selection.

Version added

2.0

Syntax

object.**Delete**

object	Required. An expression that returns an object in the **Applies to** list.

Delete method (Layer object)

See also Example

Deletes a **Layer** object. Can also delete shapes assigned to the deleted layer.

Version added

2.0

Syntax

object.**Delete** *fDeleteShapes*

object	Required. An expression that returns the **Layer** object to delete.
fDeleteShapes	Required **Integer**. 1 (**True**) to delete shapes assigned to the layer; otherwise, 0 (**False**).

Remarks

When *fDeleteShapes* is non-zero, shapes assigned only to the deleted layer are deleted. Otherwise, the shapes are simply no longer assigned to that layer.

Delete method (Page object)

See also Example

Deletes a **Page** object. Can also renumber remaining pages.

Version added

2.0

Syntax

object.**Delete** *fRenumberPages*

object	Required. An expression that returns the **Page** object to delete.
fRenumberPages	Required **Integer**. 1 (**True**) to renumber remaining pages; otherwise, 0 (**False**).

Remarks

When *fRenumberPages* is non-zero, the remaining pages' default page names are renumbered after the page is deleted, otherwise, the pages retain their names.

DeleteRow method

Deletes a row from a section in a ShapeSheet spreadsheet.

Version added

2.0

Syntax

object.**DeleteRow** *section*, *row*

object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The index of the section that contains the row.
row	Required **Integer**. The index of the row to delete.

Remarks

To remove one row at a time from a ShapeSheet section, use the **DeleteRow** method. If the section has indexed rows, the rows following the deleted row shift position. If the row does not exist, nothing is deleted.

You should not delete rows that define fundamental characteristics of a shape, such as the 1-D Endpoints row (**visRowXForm1D**) or the component row (**visRowComponent**) or the MoveTo row (**visRowVertex** + 0) in a Geometry section. You cannot delete rows from sections represented by **visSectionCharacter**, **visSectionParagraph**, and **visSectionTab**.

DeleteSection method

Deletes a ShapeSheet section.

Version added

2.0

Syntax

object.**DeleteSection** *section*

object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The index of the section to delete.

Remarks

When you delete a ShapeSheet section, all rows in the section are automatically deleted. If the specified section does not exist, nothing is deleted and no error is generated.

If a Geometry section is deleted, any subsequent Geometry sections shift up because they are indexed and no gaps can exist in an indexed range.

You can delete any section except the section represented by **visSectionObject** (although you can delete rows within that section).

For a list of section index values, see the **AddSection** method or view the Visio type library for the members of **visSectionIndices**.

DeleteSolutionXMLElement method

Example

Deletes the named SolutionXML element.

Version added

2002

Syntax

object.**DeleteSolutionXMLElement** *elementName*

object	Required. An expression that returns a **Document** object.
elementName	Required **String**. Name of the SolutionXML element to delete.

Remarks

The *elementName* argument is case-sensitive and should match the name passed as an argument to the **SolutionXMLElement** property.

If Microsoft Visio 2002 files are saved in Visio 5.0 format, SolutionXML elements are deleted. If version 2002 files are saved in Visio 2000 format, SolutionXML elements are saved but the data is inaccessible.

DeselectAll method

Example

Deselects all shapes in a window or selection.

Version added

2.0

Syntax

object.**DeselectAll**

object	Required. An expression that returns a **Window** or **Selection** object.

DockedStencils method

Returns the names of all stencils docked in a Microsoft Visio drawing window.

Version added

4.5

Syntax

object.**DockedStencils** *nameArray*

object	Required. An expression that returns a **Window** object.
nameArray	Required **String**. Array that receives the names of stencils docked in a window.

Remarks

The **DockedStencils** method returns an array of strings—the names of the stencils shown in the docked stencil panes of a window. When the window is a drawing window, the number of docked stencil panes (*n*) is equal to or greater than zero, and *n* is zero when the window isn't a drawing window.

If the **DockedStencils** method succeeds, *nameArray* returns a one-dimensional array of *n* strings indexed from zero (0) to *n* - 1. The *nameArray* argument is an out argument that is allocated by the **DockedStencils** method, ownership of which is passed back to the caller. The caller should eventually perform the **SafeArrayDestroy** procedure on the returned array. Note that the **SafeArrayDestroy** procedure has the side effect of freeing the strings referenced by the array's entries. The **DockedStencils** method fails if *nameArray* is null. (Microsoft Visual Basic and Visual Basic for Applications take care of destroying the array for you.)

If *si* is the string returned by *nameArray(i)*, then **Documents.Item**(*si*) succeeds and returns a **Document** object representing the stencil.

DoCmd method

See also

Performs the command with the indicated command ID.

Version added

4.0

Syntax

object.**DoCmd** *(intExpression)*

object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. The command to perform.

Remarks

Constants for Visio command IDs are declared by the Visio type library in **VisUICmds** and are prefixed with **visCmd**.

The **DoCmd** method works best with commands that display dialog boxes.

DrawBezier method

Creates a new shape whose path is defined by the supplied sequence of Bezier control points.

Version added

4.1

Syntax

objRet = *object*.**DrawBezier**(*xyArray*, *degree*, *flags*)

objRet	The new **Shape** object.
object	Required. The page, master, or group in which to draw the shape.
xyArray	Required **Double**. An array of alternating *x* and *y* values that define the Bezier control points for the new shape.
degree	Required **Integer**. The degree of the Bezier curve.
flags	Required **Integer**. Flags that influence how the shape is drawn.

Remarks

The *xyArray* and *degree* parameters must meet the following conditions:

1 <= *degree* <= 9

The number of points must be *k* * *degree* + 1, where *k* is a positive integer. If the first point is called *p*0, then for any integer *m* between 1 and *k*, *p*(*m* * *degree*) is assumed to be the last control point of a Bezier segment, as well as the first control point of the next.

The result is a composite curve that consists of *k* Bezier segments. The input points from *xyArray* define the curve's control points. If you want a smooth curve, make sure the points *p*(*n* - 1), *pn* and *p*(*n* + 1) are co-linear whenever *n* = *m* * *degree* with an integer *m*. The composite Bezier curve is represented in the application as a B-spline with integer *knots* of *multiplicity* = *degree*.

The control points should be in internal drawing units (inches) with respect to the coordinate space of the page, master, or group where the shape is being dropped. The passed array should be a type SAFEARRAY of 8-byte floating point values passed by reference (VT_R8|VT_ARRAY|VT_BYREF). This is how Microsoft Visual Basic passes arrays to Automation objects.

The *flags* argument is a bit mask that specifies options for drawing the new shape. Its value should be zero (0) or **visSpline1D** (8).

If *flags* is **visSpline1D** and the first and last points in *xyArray* don't coincide, the **DrawBezier** method produces a shape with one-dimensional (1-D) behavior; otherwise, it produces a shape with two-dimensional (2-D) behavior.

If the first and last points in *xyArray* do coincide, the **DrawBezier** method produces a filled shape.

DrawLine method

Adds a line to the **Shapes** collection of a page, master, or group.

Version added

2.0

Syntax

objRet = *object*.**DrawLine** (*x1*, *y1*, *x2*, *y2*)

objRet	A **Shape** object that represents the new line.
object	Required. An expression that returns a **Page**, **Master**, or **Shape** object on which to draw the line.
x1	Required **Double**. The *x*-coordinate of the line's begin point.
y1	Required **Double**. The *y*-coordinate of the line's begin point.
x2	Required **Double**. The *x*-coordinate of the line's end point.
y2	Required **Double**. The *y*-coordinate of the line's end point.

Remarks

Using the **DrawLine** method is equivalent to using the **Line** tool in Visio. The arguments are in internal drawing units with respect to the coordinate space of the page, master, or group where the line is being placed.

DrawNURBS Method

Example

Creates a new shape whose path consists of a single NURBS (nonuniform rational B-spline) segment.

Version added

2000

Syntax

objRet = *object*.**DrawNURBS**(*degree*, *xyArray*, *knots*, *flag*

objRet	An object that represents the new NURBS curve.
object	Required. An expression that returns a **Page**, **Master**, or **Sha** which to draw the shape.
degree	Required **Integer**. The spline's degree; an integer between 1 a
xyArray	Required **Double**. An array of alternating *x* and *y* values that control points coordinates; use internal drawing units (inches)
knots	Required **Double**. An array of knots.
flags	Required **Integer**. Flags that influence how the shape is draw
weights	Optional **Variant**. An array of weights.

Remarks

The **DrawNURBS** method creates a new shape whose path consists of a single NURBS segment as specified by the arguments.

The control points should be in internal drawing units (inches) with respect to the coordinate space of the page, master, or group where the shape is being dropped. The *xyArray*, *knots*, and *weights* arrays should be of type SAFEARRAY of 8-byte floating point values passed by reference (VT_R8|VT_ARRAY|VT_BYREF). This is how Microsoft Visual Basic passes arrays to Automation objects.

The *knots* argument is unit-less. The sequence of *knots* should be non-decreasing. In other words, *knots*(*i* + 1) < *knots*(*i*) is not acceptable. *knots*(*i* + 1) = *knots*(*i*) is permitted, and then the value is repeated, but the following restrictions apply:

The first knot may not be repeated more than *degree* + 1 times.

The last knot may not be repeated.

Any knot between the first and last may not be repeated more than *degree* times.

If the first knot is repeated less than *degree*+1 times, the spline is *periodic*.

The list of weights is optional. Its absence signals that the spline is *non-rational*. Weights are unit-less.

The following rules apply to the sizes of the lists. For a spline with *n* control points:

If the spline is periodic, then *n* > 2. Otherwise, *n* > *degree*.

The size of *xyArray* is 2*n*.

The size of the *weights* array is *n* (if present).

The size of the *knots* array is *n* + 1.

The conventional non-periodic spline requires *n* + *degree* + 1 knots, but the application implies the repeated knots at the end. For example, the *degree* 2 knot list (0,0,0,2,5,8) is interpreted in the application as the conventional knot

sequence (0,0,0,2,5,8,8,8).

The *flags* parameter is a bit mask that specifies options for drawing the new shape. Its value should be either zero (0) or **visSpline1D** (8). If *flags* is **visSpline1D** and if the first and last points in *xyArray* don't coincide, the **DrawNURBS** method produces a shape with one-dimensional (1-D) behavior; otherwise, it produces a shape with two-dimensional (2-D) behavior.

If the first and last points in *xyArray* do coincide, the **DrawNURBS** method produces a filled shape.

DrawOval method

Adds an ellipse to the **Shapes** collection of a page, master, or group.

Version added

2.0

Syntax

retVal = *object*.**DrawOval** (*x1*, *y1*, *x2*, *y2*)

retVal	A **Shape** object that represents the new ellipse.
object	Required. An expression that returns a **Page**, **Master**, or **Shape** object on which to draw the ellipse.
x1	Required **Double**. The left side of the ellipse's width-height box.
y1	Required **Double**. The top of the ellipse's width-height box.
x2	Required **Double**. The right side of the ellipse's width-height box.
y2	Required **Double**. The bottom of the ellipse's width-height box.

Remarks

Using the **DrawOval** method is equivalent to using the **Ellipse** tool in the application. The arguments are in internal drawing units with respect to the coordinate space of the page, master, or group where the ellipse is being placed.

DrawPolyline method

Creates a new shape whose path is a polyline along a given set of points.

Version added

2000

Syntax

objRet = *object*.**DrawPolyline**(*xyArray*, *flags*)

objRet	A **Shape** object that represents the new polyline.
object	Required. An expression that returns a **Page**, **Master**, or **Shape object** in which to draw the shape.
xyArray	Required **Double**. An array of alternating *x* and *y* values that defines points in the new shape's path.
flags	Required **Integer**. Flags that influence how the shape is drawn.

Remarks

The **DrawPolyline** method creates a new shape whose path consists of a

sequence of line segments and whose end points match the points specified in *xyArray*. Calling the **DrawPolyline** method is equivalent to calling the **DrawSpline** method with a tolerance of zero (0) and a flag of **visSplineAbrupt**.

The control points should be in internal drawing units (inches) with respect to the coordinate space of the page, master, or group where the shape is being dropped. The passed array should be a type SAFEARRAY of 8-byte floating point values passed by reference (VT_R8|VT_ARRAY|VT_BYREF). This is how Microsoft Visual Basic passes arrays to Automation objects.

The *flags* argument is a bit mask that specifies options for drawing the new shape. Its value can include **visPolyline1D** (8) or **visPolyarcs** (256). If *flags* includes

visPolyline1D and if the first and last points in *xyArray* don't coincide, the **DrawPolyline** method produces a shape with one-dimensional (1-D) behavior; otherwise, it produces a shape with two-dimensional (2-D) behavior.

visPolyarcs, then Visio will produce a sequence of arcs rather than a sequence of line segments; *xyArray* should specify the initial *x*,*y* point of the sequence followed by *x*,*y* bow triples. Visio will produce a shape with EllipticalArcTo rows where the bow of the arc matches the specified value.

If the first and last points in *xyArray* coincide, the **DrawPolyline** method produces a filled shape.

DrawRectangle method

Adds a rectangle to the **Shapes** collection of a page, master, or group.

Version added

2.0

Syntax

objRet = *object*.**DrawRectangle** (*x1*, *y1*, *x2*, *y2*)

objRet	A **Shape** object that represents the new rectangle.
object	Required. An expression that returns a **Page**, **Master**, or **Shape object** on which to draw the rectangle.
x1	Required **Double**. The left side of the rectangle's width-height box.
y1	Required **Double**. The top of the rectangle's width-height box.
x2	Required **Double**. The right side of the rectangle's width-height box.
y2	Required **Double**. The bottom of the rectangle's width-height box.

Remarks

Using the **DrawRectangle** method is equivalent to using the **Rectangle** tool in the application. The arguments are in internal drawing units with respect to the coordinate space of the page, master, or group where the rectangle is being placed.

DrawRegion method

See also Example

Draws a new shape that represents the region containing a given point.

Version added

2000

Syntax

objRet = *object*.**DrawRegion**(*tolerance*, *flags*, [*x*], [*y*], [*R*

objRet	A **Shape** object.
object	Required. An expression that returns a **Selection** object.
tolerance	Required **Double**. Error tolerance when determining the coir A distance expressed in internal units in the coordinate space object's containing shape; the maximum gap between paths t when constructing the boundaries of a region.
flags	Required **Integer**. A constant or integer that specifies how to
x	Optional **Variant**. *X*-coordinate in internal units in the coord **Selection** object.
y	Optional **Variant**. *Y*-coordinate in internal units in the coord **Selection** object.
ResultsMaster	Optional **Variant**. The **Master** object which the new **Shape** instance of.

Remarks

The **DrawRegion** method creates a new **Shape** object from pieces of the paths in the **Selection** object.

If both *x* and *y* are specified, the resulting shape is the smallest region that contains the point (*x*,*y*).

In the absence of either *x* or *y*, or if the point (*x*,*y*) is not contained in any region enclosed by the paths of the selected shapes, the result is the union of all the shapes that would have been created using the **Fragment** operation.

If no closed region is defined by the selected shapes, then the **DrawRegion** method returns **Nothing** and raises no exception.

The *flags* argument can be one or a combination of the following constants declared by the Visio type library in **VisDrawRegionFlags**.

Name	Value	Description
visDrawRegionDeleteInput	&H4	Delete items in selection.
visDrawRegionIncludeHidden	&H10	Include hidden geometry.
visDrawRegionIgnoreVisible	&H20	Exclude visible geometry.

If the **DrawRegion** method is passed a *ResultsMaster* of type VT_EMPTY or VT_ERROR (which is how VBA passes an unspecified optional argument), the new shape is not an instance of a master and the fill, line, and text styles of the new region are set to the document's default styles.

If the **DrawRegion** method is passed a reference to a **Master** object in *ResultsMaster* (type VT_UNKNOWN or VT_DISPATCH) , then the **DrawRegion** method instances that **Master** object and adds geometry computed given the **Selection** object.

The new **Shape** object has no text other than text already in *ResultsMaster*.

DrawSpline method

Creates a new shape whose path follows a given sequence of points.

Version added

4.1

Syntax

objRet = *object*.**DrawSpline**(*xyArray*, *tolerance*, *flags*)

objRet	A **Shape** object that represents the new spline.
object	Required. An expression that returns a **Page**, **Master**, or **Shape** object in which to draw the new shape.
xyArray	Required **Double**. An array of alternating *x* and *y* values that define points in the new shape's path.
tolerance	Required **Double**. How closely the path of the new shape must approximate the given points.
flags	Required **Integer**. Flags that influence how the shape is drawn.

Remarks

The **DrawSpline** method creates a new shape whose path falls within the given tolerance of the given array of points. To fit the given points exactly, specify a tolerance of zero (0). Typically, the **DrawSpline** method fits spline segments through the points, but it sometimes produces line or circular arc segments in the new shape.

The control points and tolerance are in internal drawing units (inches) with respect to the coordinate space of the page, master, or group where the shape is being dropped. The passed array should be a type SAFEARRAY of 8-byte floating point values passed by reference (VT_R8|VT_ARRAY|VT_BYREF). This is how Microsoft Visual Basic passes arrays to Automation objects.

The error from the points to the path of the resulting shape is roughly within tolerance. When the number of points is large, the actual error may sometimes exceed the prescribed tolerance.

The *flags* argument is a bit mask that specifies options for drawing the new shape. Its value should be a combination of zero or more of the following values.

Constant	Value
visSplinePeriodic	1(&H1)
visSplineDoCircles	2(&H2)
visSplineAbrupt	4(&H4)
visSpline1D	8(&H8)

If *flags* includes **visSplinePeriodic** and the following conditions are met, the application attempts to draw a periodic spline. Otherwise, Visio draws a non-periodic spline:

The last point must be a repetition of the first one.

If the flag **visSplineAbrupt** is included as well, the entire closed path outlined by the points must be free of abrupt changes of direction and curvature.

If *flags* includes **visSplineDoCircles**, Visio recognizes circular segments in the given array of points and generates circular arcs instead of spline rows for those segments.

If *flags* includes **visSplineAbrupt**, Visio breaks the spline whenever it detects an

abrupt change of direction or curvature in the point's trail. An abrupt change of direction is defined by three consecutive points A, B, C in the list, for which the distance between B and the line segment AC is more than twice the tolerance. The application also considers point B to be an abrupt change if one of the segments AB or BC is more than twice as long as the other. At a point where an abrupt change is detected, the application ends the current piece (line, arc, or spline) and starts a fresh one.

If *flags* includes **visSpline1D** and the first and last points in *xyArray* don't coincide, the **DrawSpline** method produces a shape with one-dimensional (1-D) behavior, otherwise, it produces a shape with two-dimensional (2-D) behavior.

If the first and last points in *xyArray* do coincide, the **DrawSpline** method produces a filled shape.

Drop method

Creates a new **Shape** or **Master** object by dropping an object onto a receiving object such as a stencil, drawing page, or group.

Version added

2.0

Syntax

objRet = *object*.**Drop**(*dropObject*, *x*, *y*)

objRet	The **Master** or **Shape** object created by dropping *dropObject*.
object	Required. An expression that returns an object in the **Applies to** list. The object to receive *dropObject*.
dropObject	Required. The object to drop. While this is typically a Visio object such as a **Master**, **Shape**, or **Selection** object; it can be any OLE object that provides an **IDataObject** interface.
x	Required **Integer**. The *x*-coordinate at which to place the center of the shape's width or PinX.
y	Required **Integer**. The *y*-coordinate at which to place the center

of the shape's height or PinY.

Remarks

Using the **Drop** method is similar to dragging and dropping a shape with the mouse. The object dropped (*dropObject*) can be a master or a shape on the drawing page.

To add a shape to a group or on a drawing page, apply the **Drop** method to a **Shape** or **Page** object, respectively. The center of the shape's width-height box is positioned at the specified coordinates, and a **Shape** object that represents the shape that is created is returned. When applying this method to a **Shape** object, make sure that the **Shape** object represents a group.

If *dropObject* is a **Master**, the pin of the master is dropped at the specified coordinates. A master's pin is often, but not necessarily, at its center of rotation.

To create a new master in a stencil, apply the **Drop** method to a **Document** object that represents a stencil (the stencil must be opened as an original or a copy rather than read-only). In this case, the *x* and *y* arguments are ignored, and the new master that is created is returned.

DropMany[U] method

Creates one or more new **Shape** objects on a page, in a master, or in a group. It returns an array of the IDs of the **Shape** objects it produces.

Version added

4.5

Syntax

intRet = *object*.**DropMany**(*ObjectsToInstance*, *xyArray*, *I*

intRet	**Integer**. Number of entries in *xyArray* that processed success
object	Required. An expression that returns a **Page**, **Master**, or **Shap** which to create new shapes.
ObjectsToInstance	Required **Variant**. Identifies masters or other objects from wh shapes.
xyArray	Required **Double**. An array of alternating *x* and *y* values speci positions for the new shapes.
IDArray	Required **Integer**. An array that returns the IDs of the created

Remarks

Using the **DropMany** method is like using the **Page**, **Master**, or **Shape** object's **Drop** method, except you can use the **DropMany** method to create many new **Shape** objects at once, rather than one per method call. The **DropMany** method creates new **Shape** objects on the page, in the master, or in the group shape to which it is applied (this shape is called the "target object" in the following discussion).

ObjectsToInstance should be a one-dimensional array of *n* >= 1 variants. Its entries identify objects from which you want to make new **Shape** objects. An entry often refers to a Visio application **Master** object. It might also refer to a Visio application **Shape** object, **Selection** object, or even an object from another application. The application doesn't care what the lower and upper array bounds of the *ObjectsToInstance* entries are. Call these *vlb* and *vub*, respectively.

If *ObjectsToInstance(i)* is an unknown or dispatch (in Microsoft Visual Basic for Applications, a reference to a selection, shape, master, guide, or OLE object), then the object it is referencing is instanced. This is essentially equivalent to calling **Drop**(*ObjectsToInstance*(*i*),*x*,*y*).

If *ObjectsToInstance(i)* is the integer *j*, then an instance of the **Master** object in the document stencil of the target object's document whose 1-based index is *j* is made. The EventDrop cell in the Events section of the new shape is not triggered. Use the **Drop** method instead if you want the EventDrop cell to trigger.

If *ObjectsToInstance(i)* is the string *s* (or a reference to the string *s*), then an instance of the **Master** object with name *s* in the document stencil of the target object's document is made; *s* can equal either the **Master** object's **UniqueID** or **Name** property. The EventDrop cell in the Events section of the new shape is not triggered. Use the **Drop** method instead if you want the EventDrop cell to trigger.

For *vlb* < *i* <= *vub*, if *ObjectsToInstance(i)* is empty (**Nothing** or uninitialized in Microsoft Visual Basic), then entry *i* will cause *ObjectsToInstance(j)* to be instanced again, where *j* is the largest value < *i* such that *ObjectsToInstance(j)* isn't empty. If you want to make *n* instances of the same thing, only *ObjectsToInstance(vlb)* needs to be provided.

The *xyArray* argument should be a one-dimensional array of 2*m* doubles with lowerbound *xylb* and upper bound *xyub*, where *m* >= *n*. The values in the array tell the **DropMany** method where to position the **Shape** objects it produces. *ObjectsToInstance(vlb + (i - 1))* is dropped at (*xy*[(*i* - 1)2 + *xylb*],*xy*[(*i* - 1)2 + *xylb* + 1]) for 1 <= *i* <=*n*.

Note that *m* > *n* is allowed. For *n* < *i* <= *m*, the *i*'th thing instanced is the same thing as the *n*'th thing instanced. Thus to make *m* >= 1 instances of the same thing, you can pass an *ObjectsToInstance* array with one entry and an *m* entry *xyArray* array.

If the entity being instanced is a master, the pin of the new **Shape** object is positioned at the given *xy*. Otherwise, the center of the **Shape** objects is positioned at the given *xy*.

The value *intRet* returned by the **DropMany** method is the number of *xy* entries in *xyArray* that the **DropMany** method successfully processed. If all entries processed successfully, then *m* is returned. If some entries are successfully processed prior to an error occurring, then the produced **Shape** objects are not deleted and this raises an exception yet still returns a positive *intRet*.

Presuming all *m xy* entries process correctly, the number of new **Shape** objects produced by the **DropMany** method is usually equal to *m*. In rare cases (for example, if a **Selection** object gets instanced), more than *m* **Shape** objects may be produced. The caller can determine the number of produced **Shape** objects by comparing the number of shapes in the target object before and after the **DropMany** method executes. The caller can assert the new **Shape** objects are those with the highest indices in the target object's **Shapes** collection.

If the **DropMany** method returns zero (0), *IDArray* returns **Null** (**Nothing**). Otherwise, it returns a one-dimensional array of *m* integers indexed from 0 to *m* - 1. *IDArray* is an out argument that is allocated by the **DropMany** method and ownership is passed to the program that called the **DropMany** method. The caller should eventually perform the **SafeArrayDestroy** procedure on the returned array. (Visual Basic and Visual Basic for Applications take care of this for you.)

If *IDArray* returns non-**Null** (not **Nothing**), then *IDArray*(*i* - 1), 1 <= *i* <= *intRet*, returns the ID of the **Shape** object produced by the *i*'th *xyArray* entry, provided

the *i*'th *xyArray* entry produced exactly one **Shape** object. If the *i*'th *xyArray* entry produced multiple **Shape** objects, then -1 is returned in the entry. All entries *i*, *intRet* <= *i* < *m*, return -1.

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **DropMany** method to drop more than one shape when using local names to identify the shapes. Use the **DropManyU** method to drop more than one shape when using universal names to identify the shapes.

Duplicate method

Duplicates an object or selection.

Version added

2.0

Syntax

object.**Duplicate**

object	Required. An expression that returns a **Selection** or **Shape** object.

Remarks

The **Duplicate** method duplicates the specified object or selection and adds a copy to the same page as the original. Using the **Duplicate** method is equivalent to clicking **Duplicate** on the **Edit** menu.

When used with a **Shape** object, the **Duplicate** method duplicates the shape.

When used with a **Selection** object, the **Duplicate** method duplicates the selection.

EndUndoScope method

Ends or cancels a transaction with a unique scope.

Version added

2000

Syntax

object.**EndUndoScope** (*nScopeID*, *bCommit*)

object	Required. An expression that returns an **Application** object.
nScopeID	Required **Long**. The ID of the scope to close.
bCommit	Required **Boolean**. A flag indicating that the changes made during the scope should be accepted (**True**) or canceled (**False**).

Remarks

If you need to know whether events you receive are the result of a particular operation that you initiated, use the **BeginUndoScope** and **EndUndoScope** methods to wrap your operation. In your event handlers, use the **IsInScope** property to test whether the scope ID returned by the **BeginUndoScope** method

is part of the current context. Make sure you clear the scope ID you stored from the **BeginUndoScope** property when you receive the **ExitScope** event with that ID.

You must balance calls to the **BeginUndoScope** method with calls to the **EndUndoScope** method. If you call the **BeginUndoScope** method, you should call the **EndUndoScope** method as soon as you are done with the actions that constitute your scope. Also, while actions to multiple documents should be robust within a single scope, closing a document may have the side effect of purging the undo information for the currently open scope as well as purging the undo and redo stacks. If that happens, passing *bCommit* = **False** to **EndUndoScope** does not restore the undo information.

You can also use the **BeginUndoScope** and **EndUndoScope** methods to add an action defined by an add-on to the Visio undo stream. This is useful when you are operating from modeless scenarios where the initiating agent is part of an add-on's user interface or a modeless programmatic action.

Note Most Visio actions are already wrapped in internal undo scopes, so add-ons running within the application do not need to call this method.

EnumDirectories method

Returns an array naming the folders Microsoft Visio would search given a list of paths.

Version added

4.5

Syntax

object.**EnumDirectories** *pathList*,*nameArray*

object	Required. An expression that returns an **Application** object.
pathList	Required **String**. A string of full or partial paths separated by semicolons.
nameArray	**String**. Array that receives the enumerated folder names.

Remarks

Several Visio properties such as **AddonPaths** and **TemplatePaths** accept and receive a string interpreted to be a list of path (folder) names separated by semicolons. Non-fully qualified names in the list are appended to the folder that

contains the Visio program files (*appObj.Path*). When the application looks for items in the named paths, it looks in the folders and all their subfolders.

Suppose d:\Add-ons is a path that exists and e:\Add-ons is a path that doesn't exist. If the Visio executable file is installed in c:\Visio, and AddonPaths is "Add-ons;d:\Add-ons", the application looks for add-ons in c:\Visio\Add-ons, d:\Add-ons, and any of their subfolders.

The purpose of the **EnumDirectories** method is to accept a string such as one that the **AddonPaths** property might produce and return a list of the folders that the application enumerates when processing such a string.

If the **EnumDirectories** property succeeds, *nameArray* returns a one-dimensional array of *n* strings indexed from 0 to *n* - 1. Each string is the fully qualified name of a folder that exists. The list names those folders designated in the path list that exist and all their subfolders.

The *nameArray* argument is an out argument that is allocated by the **EnumDirectories** method and ownership is passed back to the caller. The caller should eventually perform the **SafeArrayDestroy** procedure on the returned array. (Visual Basic and Visual Basic for Applications automatically free the strings referenced by the array's entries.)

ExecuteLine method

Example

Executes a line of Microsoft Visual Basic code.

Version added

4.5

Syntax

```
object.ExecuteLine stringExpression
```

object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. A string that will be interpreted as Microsoft Visual Basic for Applications (VBA) code.

Remarks

The VBA project of the **Document** object is told to execute the supplied string. VBA treats the string as it would treat the same string typed into its Immediate window.

The **ExecuteLine** method creates a VBA project in the document if one does not

exist.

Here are some possibilities:

```
ThisDocument.ExecuteLine("SomeMacro")
    'Executes the macro (argumentless procedure) named S
    'that is in some module of the VBA project of ThisDocu

ThisDocument.ExecuteLine("SomeProc 1, 2, 3")
    'Executes the procedure named SomeProc and passes it

ThisDocument.ExecuteLine("Module1.SomeProc 1, 2, 3"
    'Same as previous example, but procedure name qualifi
    'with module name.

ThisDocument.ExecuteLine("UserForm1.Show")
    'Shows the form UserForm1.

ThisDocument.ExecuteLine("Debug.Print ""some string""
    'Prints "some string" to VBA's Immediate window.

ThisDocument.ExecuteLine("Debug.Print Documents.Cou
    'Prints number of open documents to Immediate windo

ThisDocument.ExecuteLine("ThisDocument.Save")
    'Tells ThisDocument to save itself.
```

Export method

Example

Exports an object from Microsoft Visio to a file format such as .pcx, .eps, or .htm.

Version added

3.0

Syntax

object.**Export** *fileName*

object	Required. An expression that returns a **Page**, **Master**, **Selection**, or **Shape** object to export.
fileName	Required **String**. The fully qualified path and name of the file to receive the exported object.

Remarks

The file extension indicates which export filter to use. If the filter is not installed, the **Export** method returns an error. The **Export** method uses the default preference settings for the specified filter and does not prompt the user for non-

default arguments.

The **Export** method of a **Page** object supports saving to HTML file format using the extension .htm or .html. Pages are exported using the settings that were last selected in the **Save As** dialog box.

If the specified file already exists, it is replaced without prompting the user.

ExportIcon method

Example

Exports the icon for a **Master** object to a named file or the Clipboard.

Version added

4.5

Syntax

object.**ExportIcon** *fileName*, *flags*, [*TransparentRGB*]

object	Required. An expression that returns a **Master** object.
fileName	Required **String**. The file to which to export the icon.
flags	Required **Integer**. The format in which to write the exported file.
TransparentRGB	Optional **Variant**. The color to substitute for any transparent areas of the exported icon image.

Remarks

If *fileName* is empty, the master's icon is copied to the Clipboard.

If the value of *flags* is **visIconFormatVisio** (0), the icon is exported in the application internal icon format. The **ImportIcon** method accepts files written in this format.

If the value of *flags* is **visIconFormatBMP** (2), the icon is exported in bitmap (.bmp) file format.

Starting with Visio 2000, you can use the *TransparentRGB* argument with the **ExportIcon** method. If *TransparentRGB* is omitted, the color defaults to black, which simulates Visio 5.0 behavior.

FitCurve method

Example

Reduces the number of geometry segments in a shape or shapes by replacing them with similar spline, arc, and line segments that approximate the paths of the initial segments. Typically, this reduces the number of segments in the shape.

Version added

4.1

Syntax

object.**FitCurve** *tolerance*, *flags*

object	Required. An expression that returns a **Shape** or **Selection** object whose path is to be replaced.
tolerance	Required **Double**. How closely the resulting paths must match the shape's original paths.
flags	Required **Integer**. Flags that influence how the shape is drawn.

Remarks

The **FitCurve** method of a **Selection** object optimizes each of the shapes in the

selection. It does not combine the selected shapes into a single shape.

The paths resulting from the **FitCurve** method fall within the given tolerance of the initial paths. Tolerance should be in internal drawing units (inches). To match the initial paths exactly, specify a tolerance of zero (0).

The *flags* argument is a bit mask that specifies options for optimizing the paths. Its value should be a combination of zero or more of the following values.

Constant	Value	Description
visSplinePeriodic	&H1	Produce periodic splines if appropriate.
visSplineDoCircles	&H2	Recognize circular segments in the shape(s) and generate circular arcs instead of spline rows for those segments.
visSplineAbrupt	&H4	Break the resulting splines whenever an abrupt change of direction or curvature in a path is detected.

FlipHorizontal method

Example

Flips an object horizontally.

Version added

2.0

Syntax

object.**FlipHorizontal**

object	Required. An expression that returns a **Shape** or **Selection** object to flip.

FlipVertical method

Example

Flips an object vertically.

Version added

2.0

Syntax

object.**FlipVertical**

object	Required. An expression that returns a **Shape** or **Selection** object to flip.

Follow method

Causes Microsoft Visio to navigate to a hyperlink.

Version added

5.0

Syntax

object.**Follow**

object	Required. An expression that returns a **Hyperlink** object.

FollowHyperlink method

Example

Navigates to an arbitrary document-based hyperlink.

Version added

5.0

Syntax

object.**FollowHyperlink** (*Address*, *SubAddress*, [*ExtraInf*

object	Required. An expression that returns a **Document**
Address	Required **String**. The address to which you want t
SubAddress	Required **String**. The subaddress to which you wa string.
ExtraInfo	Optional **Variant**. Extra URL request information
Frame	Optional **Variant**. The HTML frame to which to n
NewWindow	Optional **Variant**. Specifies if a new window is to
res1	Optional. Unused.
res2	Optional. Unused.

res3	Optional. Unused.

Remarks

From Microsoft Visual Basic or Visual Basic for Applications, do not pass a value for optional arguments. From C/C++, pass an empty variant for optional arguments.

Visio 4.5 provided an undocumented **Hyperlink** method for a **Document** object with the following signature:

```
HRESULT FollowHyperlink[in] BSTR Target, [in] BSTR
```

Visio 5.0 and later still support this method but it has been renamed to **FollowHyperlink45**:

```
HRESULT FollowHyperlink45[in] BSTR Target, [in] BST
```

FormatResult method

Formats a string or number into a string according to a format picture, using specified units for scaling and formatting.

Version added

4.5

Syntax

stringRet = *object*.**FormatResult**(*stringOrNumber*, *unitsI*

stringRet	**String**. The evaluated result formatted according to format
object	Required. An expression that returns an **Application** objec
stringOrNumber	Required **Variant**. String or number to be formatted; can b floating point number, or integer.
unitsIn	Required **Variant**. Measurement units to attribute to *string*
unitsOut	Required **Variant**. Measurement units to express the result
format	Required **String**. Picture of what the result string should lo

Remarks

If passed as a string, *stringOrNumber* might be the formula or prospective formula of a cell, or the result or prospective result of a cell expressed as a string. The **FormatResult** method evaluates the string and formats the result. Because the string is being evaluated outside the context of being the formula of a particular cell, the **FormatResult** method returns an error if the string contains any cell references.

Possible values for *stringOrNumber* include:

1.7

3

"2.5"

"4.1 cm"

"12 ft - 17 in + (12 cm / SQRT(7))"

The *unitsIn* and *unitsOut* arguments can be strings such as "inches", "inch", "in.", or "i". Strings may be used for all supported Visio units such as centimeters, meters, miles, and so on. You can also use any of the unit constants declared by the Visio type library in **VisUnitCodes**. A list of valid units is also included in About units of measure.

If *stringOrNumber* is a string, *unitsIn* specifies how to interpret the evaluated result and is only used if the result is a scalar. For example, the expression "*4 * 5 cm*" evaluates to 20 cm, which is not a scalar so *unitsIn* is ignored. The expression "*4 * 5*" evaluates to 20 which is a scalar and is interpreted using the specified *unitsIn*.

The *unitsOut* argument specifies the units in which the returned string should be expressed. If you want the results expressed in the same units as the evaluated expression, pass "NOCAST" or **visNoCast**.

format is a string that specifies a template or picture of the string produced by the **FormatResult** method. For details, see the FORMAT function. A few of the possibilities are:

: Ouput a single digit, but not if it's a leading or trailing 0.

0 : Output a single digit, even if it is a leading or trailing 0.

. : Decimal placeholder.

, : Thousands separator.

"text" or 'text' : Output enclosed text as is.

\c : Output the character c.

Examples

- Where a string is specified
- Where a number is specified

Fragment method

Example

Breaks selected shapes into smaller shapes.

Version added

2.0

Syntax

object.**Fragment**

object	Required. An expression that returns a **Selection** object that contains the shapes to fragment.

Remarks

Using the **Fragment** method is equivalent to clicking **Fragment** on the **Operations** submenu of the **Shape** menu. The produced shapes are the topmost shapes in the containing shape of the selected shapes. They inherit the formatting of the first selected shape and have no text.

The original shapes are deleted and there aren't any shapes selected when the

operation is complete.

GetFilterCommands method

Example

Returns an array of command ranges and a **True** or **False** value indicating how to filter events for that command range.

Version added

2002

Syntax

retVal = object.**GetFilterCommands()**

object	Required. An expression that returns an **Event** object.
retVal	**Long**. An array of command ranges and a **True** or **False** value specifying how to filter events for that command range.

Remarks

The event filters described in the array returned by the **GetFilterCommands** method provide developers a way of ignoring specified events based on command ID. The array returned is that passed to the **SetFilterCommands** method for this **Event** object.

The array that is returned by the **GetFilterCommands** method can be interpreted in the following manner:

The number of elements in the array is a multiple of 3, as follows:

The first element contains the beginning command ID of the range (any member of **VisUICmds**).

The second element contains the end command ID of the range (any member of **VisUICmds**).

The third element contains a **True** or **False** value, which indicates whether you are listening to events for that command range (**True** to listen to events; **False** to exclude events).

For an event to successfully pass through a command filter, it must satisfy the following criteria:

It must have a valid command ID.

If all filters are **True**, the event must match at least one filter.

If all filters are **False**, the event must not match any filter.

If the filters are a mixture of **True** and **False**, the event must match at least one **True** filter and not match any **False** filters.

If there are no **True** ranges defined in the array, events are considered **True**.

For details about defining event filters using command IDs, see the **SetFilterCommands** method.

GetFilterObjects method

Example

Returns an array of object types and a **True** or **False** value indicating how to filter events for that object.

Version added

2002

Syntax

retVal = object.**GetFilterObjects()**

object	Required. An expression that returns an **Event** object.
retVal	**Long**. An array of objects types and a **True** or **False** value specifying how to filter events for that object.

Remarks

The event filters described in the array returned by the **GetFilterObjects** method provide developers a way of ignoring specified events based on object type. The array returned is that passed to the **SetFilterObjects** method for this **Event** object.

The array that is returned by the **GetFilterObjects** method can be interpreted in the following manner.

The number of elements in the array is a multiple of 2:

The first element contains an object type (one of **visTypePage**, **visTypeGroup**, **visTypeShape**, **visTypeForeignObject**, **visTypeGuide**, or **visTypeDoc**).

The second element contains a **True** or **False** value indicating whether you are listening to events for that object (**True** to listen to an object's events; **False** to exclude an object's events).

For an event to successfully pass through an object event filter, it must satisfy the following criteria:

It must be a valid object type.

If all filters are **True**, the event must match at least one filter.

If all filters are **False**, the event must not match any filter.

If the filters are a mixture of **True** and **False**, the event must match at least one **True** filter and not match any **False** filters.

If there are no **True** ranges defined in the array, events are considered **True**.

For details about defining event filters using command IDs, see the **SetFilterObjects** method.

GetFilterSRC method

Example

Returns an array of cell ranges and a **True** or **False** value indicating whether you are filtering events for that range.

Version added

2002

Syntax

retVal = object.**GetFilterSRC()**

object	Required. An expression that returns an **Event** object.
retVal	**Integer**. An array of cell ranges and a **True** or **False** value specifying how to filter events for that range.

Remarks

The event filters described in the array returned by the **GetFilterSRC** method provide developers a way of ignoring specified events based on object type. The array returned is that passed to the **SetFilterSRC** method for this **Event** object.

The array that is returned by the **GetFilterSRC** method can be interpreted in the following manner.

The number of elements in the array is a multiple of 7. These seven elements contain the following values:

The first three elements describe the section, row, and cell of the beginning cell of the range.

The next three elements describe the section, row, and cell of the end cell of the range.

The last element contains a **True** or **False** value indicating whether you want to receive events for the specified range of cells (**True** to listen to events for a range of cells; **False** to exclude events for the range of cells).

For an event to successfully pass through a cell range filter, it must satisfy the following criteria:

It must be a valid section, row, cell reference.

If all filters are **True**, the event must match at least one filter.

If all filters are **False**, the event must not match any filter.

If the filters are a mixture of **True** and **False**, the event must match at least one **True** filter and not match any **False** filters.

If there are no **True** ranges defined in the array, events are considered **True**.

For details about defining event filters using command IDs, see the **SetFilterSRC** method.

GetFormulas[U] method

Returns the formulas of many cells.

Version added

4.5

Syntax

object.**GetFormulas** *SID_SRCStream*, *formulas*

object	Required. An expression that returns a **Page**, **Master**, **Shape**, or **Style** object.
SID_SRCStream	Required **Integer**. Stream identifying cells to be queried.
formulas	Required **Variant**. Array that receives formulas of queried cells.

Remarks

The **GetFormulas** method is like the **Formula** method of a **Cell** object, except you can use it to obtain the formulas of many cells at once, rather than one cell at a time. The **GetFormulas** method is a specialization of the **GetResults** method,

which can be used to obtain cell formulas or results. Setting up a call to the **GetFormulas** method involves slightly less work than setting up the **GetResults** method.

For **Shape** or **Style** objects you can use the **GetFormulas** method to get formulas of any set of cells.

For a **Page** or **Master** object you can use the **GetFormulas** method to get formulas of any set of cells in any set of shapes of the page or master.

SID_SRCStream is an array of 2-byte integers:

For **Shape** or **Style** objects, *SID_SRCStream* should be a one-dimensional array of 3*n* 2-byte integers for some *n* >= 1. **GetFormulas** interprets the stream as:

{ sectionIdx, rowIdx, cellIdx }n

where *sectionIdx* is the section index of the desired cell, *rowIdx* is its row index and *cellIdx* is its cell index.

For **Page** or **Master** objects, *SID_SRCStream* should be a one-dimensional array of 4*n* 2-byte integers for *n* >= 1. The **GetFormulas** method interprets *SID_SRCStream* as:

{ sheetID, sectionIdx, rowIdx, cellIdx }n

where *sheetID* is the **ID** property of the **Shape** object on the page or master whose cell formula is desired.

Note If the *sheetID* in an entry is **visInvalShapeID** (-1) or if the bottom byte of *sectionIdx* is **visSectionInval** (255), then the entry will be ignored and an empty variant will be returned in the corresponding results array entry. The motivation for this is that the same *SID_SRCStream* array can be used on several calls to **GetFormulas**, **SetFormulas**, and similar methods with the caller only needing to make minor changes to the stream between calls.

If the **GetFormulas** method succeeds, *formulas* returns a one-dimensional array of *n* variants indexed from 0 to *n* - 1. Each variant returns a formula as a string. *Formulas* is an out argument that is allocated by the **GetFormulas** method, which passes ownership back to the caller. The caller should eventually perform

the **SafeArrayDestroy** procedure on the returned array. Note that the **SafeArrayDestroy** procedure has the side effect of clearing the variants referenced by the array's entries, hence deallocating any strings the **GetFormulas** method returns. (Microsoft Visual Basic and Visual Basic for Applications take care of this for you.) The **GetFormulas** method fails if *formulas* is **Null**.

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **GetFormulas** method to get more than one formula using local syntax. Use the **GetFormulasU** method to get more than one formula using universal syntax.

GetNames[U] method

See also

Returns the names of all items in a **Documents**, **Pages**, **Masters**, **Styles**, or **Addons** collection.

Version added

4.5

Syntax

object.**GetNames** *nameArray*

object	Required. An expression that returns a collection from the **Applies to** list.
nameArray	Required **String**. Array that receives names of members of the indicated object.

Remarks

If the **GetNames** method succeeds, *nameArray* returns a one-dimensional array of *n* strings indexed from 0 to *n* - 1, where *n* equals the **Count** property of the object. *nameArray* is an out argument that is allocated by the **GetNames**

method, which passes ownership back to the caller. The caller should eventually perform the **SafeArrayDestroy** procedure on the returned array. Note that the **SafeArrayDestroy** procedure has the side effect of freeing the strings referenced by the array's entries. The **GetNames** method fails if called with *!nameArray or nameArray*. (Microsoft Visual Basic and Visual Basic for Applications take care of this for you.)

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **GetNames** method to get more than one object's local name. Use the **GetNamesU** method to get more than one object's universal name.

GetPolylineData method

Example

Returns the points recorded in a polyline row.

Version added

2000

Syntax

object.**GetPolylineData** *flags*, *xyArray*

object	Required. An expression that returns a **Row** object.
flags	Required **Integer**. Flags that influence the points returned.
xyArray	Required **Double**. Returns an array of alternating *x* and *y* values specifying the points recorded in the row.

Remarks

If the row's type is not **visTagPolylineTo**, an exception is raised.

If the **GetPolylineData** method succeeds, *xyArray* returns a one-dimensional array of *n* doubles (VT_R8) indexed from 0 to *n* - 1. The argument *xyArray* is an

out argument that is allocated by the **GetPolylineData** method, which passes ownership back to the caller. The caller should eventually perform **SafeArrayDestroy** on the returned array. (Microsoft Visual Basic and Visual Basic for Applications manage this for you.)

The *flags* argument is a bit mask that specifies options for returning points. Its value should be a combination of zero or more of the following values.

Constant	Value	Description
visGeomExcludeLastPoint	&H1	The last point of the polyline (the X and Y cells in the row) will not be included in *xyArray.*
visGeomWHPct	&H10	The values returned in *xyArray* will be percentages of width/height.
visGeomXYLocal	&H20	The values returned in *xyArray* will be local, internal units in the drawing.

GetResults method

Gets the results or formulas of many cells.

Version added

4.5

Syntax

object.**GetResults** *SID_SRCStream*, *flags*, *units*, *results*

object	Required. An expression that returns a **Page**, **Master**, **Shape**, or **Style** object.
SID_SRCStream	Required **Integer**. Array identifying cells to be queried.
flags	Required **Integer**. Flags that influence the type of entries returned in results.
units	Required **Variant**. Array of measurement units that results are to be returned in.
results	Required **Variant**. Array that receives results or formulas of queried cells.

Remarks

The **GetResults** method is like the **Result** method for the **Cell** object, except that it can be used to get the results (values) of many cells at once, rather than one cell at a time.

For **Shape** or **Style** objects, you can use the **GetResults** method to get results of any set of cells.

For a **Page** or **Master** object, you can use the **GetResults** method to get results of any set of cells in any set of shapes of the page or master.

SID_SRCStream is an array of 2-byte integers:

For **Shape** or **Style** objects, *SID_SRCStream* should be a one-dimensional array of 3*n* 2-byte integers for *n* >= 1. The **GetResults** method interprets *SID_SRCStream* as:

{ sectionIdx, rowIdx, cellIdx }n

where *sectionIdx* is the section index of the desired cell, *rowIdx* is its row index and *cellIdx* is its cell index.

For **Page** or **Master** objects, *SID_SRCStream* should be a one-dimensional array of 4*n* 2-byte integers for *n* >= 1. The **GetResults** method interprets *SID_SRCStream* as:

{ sheetID, sectionIdx, rowIdx, cellIdx }n

where *sheetID* is the **ID** property of the **Shape** object on the page or master whose cell result is desired.

Note If the *sheetID* in an entry is **visInvalShapeID** (-1) or if the bottom byte of *sectionIdx* is **visSectionInval** (255), then the entry will be ignored and an empty variant will be returned in the corresponding results array entry. The motivation for this is that the same *SID_SRCStream* array can be used on several calls to **GetResults**, **SetResults**, and similar methods with the caller only needing to make minor changes to the stream between calls.

The *flags* argument indicates what data type the returned results should be

expressed in. Its value should be one of the following.

Constant	Value	Description
visGetFloats	0	Results returned as doubles (VT_R8).
visGetTruncatedInts	1	Results returned as truncated long integers (VT_I4).
visGetRoundedInts	2	Results returned as rounded long integers (VT_I4).
visGetStrings	3	Results returned as strings (VT_BSTR).
visGetFormulas	4	Formulas returned as strings (VT_BSTR).
visGetFormulasU	5	Formulas returned in universal syntax (VT_BSTR).

The *units* argument is an array that controls what measurement units individual results are returned in. Each entry in the array can be a string such as "inches", "inch", "in.", or "i". Strings may be used for all supported Visio units such as centimeters, meters, miles, and so on. You can also indicate desired units with integer constants (**visCentimeters**, **visInches**, etc.) declared by the Visio type library. Note that the values specified in the *units* array have no effect if *flags* is **visGetFormulas**.

If not null, the application expects *units* to be a one-dimensional array of 1 <= *u* **Variants**. Each entry can be a string or integer code, or empty (nothing). If the *i*'th entry is empty, then the *i*'th returned result is returned in the units designated by *units*(*j*), where *j* is the index of the most recent prior non-empty entry. Thus if you want all returned values to be in the same units, you need only pass a *units* array with one entry. If there is no prior non-empty entry, or if no *units* array is supplied, then **visNumber** (0x20) is used. This causes internal units (like the **ResultIU** property of a **Cell** object) to be returned.

If the **GetResults** method succeeds, *results* returns a one-dimensional array of *n* variants indexed from zero (0) to *n* - 1. The type of the returned variants is a function of *flags*. *Results* is an out argument that is allocated by the **GetResults**

method, which passes ownership back to the caller. The caller should eventually perform **SafeArrayDestroy** on the returned array. Note that **SafeArrayDestroy** has the side effect of clearing the variants referenced by the array's entries, hence deallocating any strings the **GetResults** method returns. (Microsoft Visual Basic and Visual Basic for Applications take care of this for you.)

GetViewRect method

Example

Returns the page coordinates of a window's borders.

Version added

2000

Syntax

object.**GetViewRect** (*doubleLeft*, *doubleTop*, *doubleWidth*

object	Required. An expression that returns a **Window** object.
doubleLeft	Required **Double**. The coordinate in page units of the left si
doubleTop	Required **Double**. The coordinate in page units of the top of
doubleWidth	Required **Double**. The distance in page units from the left si the window.
doubleHeight	Required **Double**. The distance in page units from the top to window.

Remarks

If the **Window** object is not a **visDrawing** type, then the **GetViewRect** method raises an exception.

GetWindowRect method

Example

Gets the size and position of the client area of a window.

Version added

2000

Syntax

object.**GetWindowRect** *pnLeft*, *pnTop*, *pnWidth*, *pnHeigh*

object	Required. An expression that returns a **Window** object.
pnLeft	Required **Long**. The coordinate of the left side of the window.
pnTop	Required **Long**. The coordinate of the top of the window.
pnWidth	Required **Long**. The distance in pixels from the left side to the right side of the window.
pnHeight	Required **Long**. The distance in pixels from the top to the bottor of the window.

Remarks

The **GetWindowRect** method gets the size and position of the client area of the window with respect to the window that owns the **Windows** collection to which it belongs. For the **Windows** collection of an **Application** object, the "with respect to" window is the MDICLIENT window of the Visio main window. For the **Windows** collection of a **Window** object, the "with respect to" window is the client area of the drawing window.

GlueTo method

Glues one shape to another, from a cell in the first shape to a cell in the second shape.

Version added

2.0

Syntax

object.**GlueTo** *gluetocell*

object	Required. An expression that returns a **Cell** object.
gluetocell	Required. An expression that returns a **Cell** object that represents the part of the shape to glue to.

Remarks

Following is a list of possible connections.

- From the begin or end cell of a 1-D shape to…

- From the edge (a cell in the Alignment section) of a 2-D shape to…
- From an outward or inward/outward connection point cell of a 1-D shape to…
- From an outward or inward/outward connection point cell of a 2-D shape to…
- From a control point cell to…

For details about connection point type and direction, see the Connection Points section.

GlueToPos method

Glues one shape to another from a cell in the first shape to an *x*,*y* position in the second shape.

Version added

2.0

Syntax

object.**GlueToPos** *shpObject*, *x*, *y*

object	Required. An expression that returns a **Cell** object.
shpObject	Required object. An expression that returns the **Shape** object to be glued to.
x	Required **Double**. The *x*-coordinate of the position to glue to.
y	Required **Double**. The *y*-coordinate of the position to glue to.

Remarks

The **GlueToPos** method creates a new connection point at the location

determined by *x* and *y*, which represent decimal fractions of the specified shape's width and height, respectively, rather than coordinates. For example, the following creates a connection point at the center of *shpObject* and glues the part of the shape that *celObj* represents to that point:

celObj.**GlueToPos** *shpObject*, *0.5*, *0.5*

Gluing the X cell of a Controls section row or a BeginX or EndX cell automatically glues the Y cell of the Controls section row or the BeginY or EndY cell, respectively. (The reverse is also true.)

Group method

Groups the objects that are selected in a selection, or it converts a shape into a group.

Version added

2.0

Syntax

shpObj = object.**Group**

shpObj	Object. The resulting **Shape** object.
object	Required. An expression that returns a **Shape** or **Selection** object.

IconFileName method

Sets a custom icon file for a menu or toolbar item.

Version added

4.0

Syntax

object.**IconFileName**("*fileString* [,*N*]")

object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** that loads the icon file.
fileString	Required **String**. The path and name of the ICO, EXE, DLL, or VSL file to load.
N	Optional **Integer**. The icon resource location. If you are using an EXE, DLL, or VSL file, *N* specifies the location of the icon in the file, beginning with zero (0).

Remarks

The **IconFileName** method loads the file that contains the icon, saves the bits,

and discards the file name.

If the icon contains multiple images, Visio always uses the 16x16 (16-color) image.

Unless *fileString* is a fully qualified path, the application searches for the ICO, EXE, DLL, or VSL file in the folders indicated by the **Application** object's **AddonPaths** property (assuming that the **UIObject** object is in the Visio process).

Import method

Example

Imports a file into the application.

Version added

3.0

Syntax

objRet = *object*.**Import**(*fileName*)

objRet	A **Shape** object that represents the new shape imported from the file.
object	Required. An expression that returns the **Page**, **Master**, or **Shape** object to receive the new shape.
fileName	Required **String**. The name of the file to import; must be a fully qualified path.

Remarks

The **Import** method imports the file specified by *fileName* onto a page, or into a master or group.

The file extension indicates which import filter to use. If the filter is not installed, the **Import** method returns an error. The **Import** method uses the default preference settings for the specified filter and does not prompt the user for non-default arguments.

ImportIcon method

Example

Imports the icon for a **Master** object from a named file.

Version added

4.5

Syntax

object.**ImportIcon** *fileName*

object	Required. An expression that returns the **Master** or **MasterShortcut** object to receive the new icon.
fileName	Required **String**. The name of the file to import.

Remarks

The **ImportIcon** method can only import files that were produced by exporting a master icon in the application's internal icon format (**visIconFormatVisio**)—it does not accept icons in other file formats.

InsertFromFile method

Example

Adds a linked or embedded object to a page, master, or group.

Version added

4.1

Syntax

objRet = *object*.**InsertFromFile**(*filename*, *flags*)

objRet	A **Shape** object representing the newly linked or embedded object.
object	Required. An expression that returns a **Page**, **Master**, or **Shape** object in which to embed or link the object.
filename	Required **String**. The name of the file that contains the object to link or embed.
flags	Required **Integer**. Flags that influence how the object is inserted.

Remarks

The **InsertFromFile** method creates a new shape that represents a linked or embedded OLE object.

The *flags* argument is a bit mask that should be a combination of the following values.

Constant	Value	Description
visInsertLink	&H8	If set, the new shape represents an OLE link to the named file. Otherwise, the **InsertFromFile** method produces an OLE object from the contents of the named file and embeds it in the document that contains the page, master, or group.
visInsertIcon	&H10	Display the new shape as an icon.

InsertObject method

Example

Adds a new embedded object or ActiveX control to a page, master, or group.

Version added

4.1

Syntax

objRet = *object*.**InsertObject**(*ClassOrProgID*, *flags*)

objRet	A **Shape** object that represents the newly created object or control.
object	Required. An expression that returns a **Page**, **Master**, or **Shape** object in which to create the object or control.
ClassOrProgID	Required **String**. Identifies the type of object or control to create.
flags	Required **Integer**. Flags that influence the operation.

Remarks

ClassOrProgID is a string that identifies the kind of object or control to create. It can be either the object or control's class ID (GUID) in string form or the object or control's program ID of the handler for the class.

If *ClassOrProgID* is a string representing a class ID, it looks like "{D3E34B21-9D75-101A-8C3D-00AA001A1652}."

If *ClassOrProgID* is a string representing a program ID, it looks like "paint.picture" or "forms.combobox.1".

See vendor-specific documentation or browse the registry to determine which class IDs and program IDs are associated with objects and controls provided by other applications.

The *flags* argument is a bit mask that can include one of the following values.

Constant	Value	Description
visInsertIcon	&H10	Display the new shape as an icon.
visInsertDontShow	&H1000	Don't execute the new object's show verb.

If both **visInsertIcon** and **visInsertDontShow** are specified, the **InsertObject** method fails. If you want to insert an object that is displayed as an icon, you must allow the application to execute the object's show verb.

The *flags* argument can also include one of the following values.

Constant	Value
visInsertAsControl	&H2000
visInsertAsEmbed	&H4000

Values in **visInsertAsControl** and **visInsertAsEmbed** only have an effect if the class identified by *ClassOrProgID* is identified in the registry as a control that can be inserted. If neither **visInsertAsControl** nor **visInsertAsEmbed** is specified and the object can be either a control or an embedded object, the application inserts it as a control.

In rare cases, Visio 5.0 or later versions may insert a control whereas earlier versions of Visio would have responded to the same call by inserting an embedded object. If a control is inserted, this method places the document in

design mode, causing any code executing in the document to halt until the document is returned to run mode.

Intersect method

Example

Creates one closed shape from the area in which selected shapes overlap or intersect.

Version added

4.0

Syntax

object.**Intersect**

object	Required. An expression that returns a **Selection** object that contains the shapes to intersect.

Remarks

The **Intersect** method is equivalent to clicking **Intersect** on the **Operations** submenu on the **Shape** menu in Visio. The produced shape will be the topmost shape in its containing shape and will inherit the text and formatting of the first selected shape.

The original shapes are deleted and no shapes are selected when the operation is complete.

InvokeHelp method

See also Example

Performs Help operations using the Microsoft Visio Help system.

Version added

2002

Syntax

object.**InvokeHelp** *bstrHelpFileName, command, data*

object	Required. An expression that returns an **Application** object.
bstrHelpFileName	Required **String**. Specifies an HTML file, a URL, a compiled HTML file, or an optional window definition (preceded with a ">" character). If the command being used does not require a file or URL, this value may be "".
command	Required **Long**. The action to perform.
data	Required **Long**. Any data that is required based on the value of the c*ommand* argument.

Remarks

Using the **InvokeHelp** method, you can create a custom Help system that is integrated with the Visio Help system. To enable your custom Help to appear in the same tiled MSO Help window as Visio Help, do not specify a window definition in the *bstrHelpFileName* argument.

The arguments passed to the **InvokeHelp** method correspond to those described

in the HTML Help API. For a list of *command* values, see the HTML Help API Reference on the Microsoft Developer Network (MSDN) Web site. Microsoft Visual Basic programmers can use the numeric equivalent of the C++ constants defined in the HTML Help API header files.

For example, use the following code to show the default Visio Help window:

Application.InvokeHelp "Visio.chm", 15, 0

Or use the following code to hide the Visio Help window:

Application.InvokeHelp "", 18, 0

For more information about the HTML Help API, search for "HTML Help API overview" on the MSDN Web site.

Join method

Example

Creates a new shape by joining selected shapes.

Version added

4.1

Syntax

object.**Join**

object	Required. An expression that returns a **Selection** object containing the shapes to join.

Remarks

The **Join** method is equivalent to clicking **Join** on the **Operation** submenu on the **Shape** menu in Visio. The new shape inherits the text and formatting of the first selected shape and is the topmost shape in its container—the *n*th shape in the **Shapes** collection of its containing shape, where *n* = Count.

The original shapes are deleted and no shapes are selected when the operation is

complete.

The **Join** method and the **Combine** method are similar but differ in the following ways:

Join coalesces abutting line and curve segments in the original shapes into a single Geometry section in the resulting shape.

Combine produces a shape that has one Geometry section for each original shape. The resulting shape has holes in regions where the original shapes overlapped.

You might want to join shapes after importing a non-Visio drawing in which apparent polylines are represented by many independent shapes, each possessing a single line or curve segment. By joining the shapes that constitute a polyline in such a drawing, you can replace many single-segment shapes with one multiple-segment shape.

Layout method

See also Example

Lays out the shapes and/or reroutes the connectors for the page, master, group, or selection.

Version added

4.5

Syntax

object.Layout

object	Required. An expression that returns a **Page**, **Master**, **Shape** or **Selection** object whose shapes are to be repositioned.

Remarks

Using the **Layout** method is equivalent to clicking **Lay Out Shapes** on the **Shape** menu.

Behavior of the **Layout** method can be influenced by setting the formulas or results of cells in the Page Layout and Shape Layout sections of the page, master, or group to be laid out. You can infer how these cells influence the behavior of the **Layout** method by examining the effect of various **Lay Out Shapes** dialog box options on the values of these cells.

To lay out a subset of the shapes of a page, master, or group, establish a **Selection** object in which the shapes to be laid out are selected, then invoke the **Layout** method. If the **Layout** method is performed on a **Selection** object and

the object has no shapes selected, all shapes in the page, master, or group of the selection are laid out.

LoadFromFile method

Loads a Microsoft Visio application **UIObject** object from a file.

Version added

4.0

Syntax

object.**LoadFromFile** *fileName*

object	Required. An expression that returns a **UIObject** object to receive data from the file.
fileName	Required **String**. The name of the file to load.

Remarks

You must use the **SaveToFile** method to save a **UIObject** object in a file that can be loaded with the **LoadToFile** method.

Open method (Documents collection)

Opens an existing file so it can be edited.

Version added

2.0

Syntax

docObjRet = *docsObj*.**Open** (*fileName*)

docObjRet	A **Document** object that represents the file that was opened.
docsObj	Required. An expression that returns the **Documents** collection to receive the opened file.
fileName	Required **String**. The name of a file to open.

Remarks

When you use the **Open** method to open a **Document** object, it opens a Visio file as an original. Depending on the file extension, the **Open** method opens a drawing (.vsd), a stencil (.vss), a template (.vst), a workspace (.vsw), an XML drawing (.vdx), an XML stencil (.vsx), or an XML template (.vtx). You can also

open and convert non–Visio files to Visio files using this method. If the file does not exist or the file name is invalid, no **Document** object is returned and an error is generated.

If a valid stencil (.vss) file name is passed, the original stencil file is opened, which means you can edit its masters. Unless you want to create or edit the masters, open a stencil as read-only through an associated template or by using the **OpenEx** method.

Open method (Master object)

Example

Opens an existing master so it can be edited.

Version added

4.1

Syntax

```
masterObjCopy = masterObj.Open
```

masterObjCopy	A temporary copy of *masterObj*.
masterObj	Required. An expression that returns a **Master** object to be edited.

Remarks

You can use the **Open** method for a **Master** object in conjunction with the **Close** method to reliably edit the shapes and cells of a master. In previous versions of Visio, you could edit a **Master** object's shapes and cells, but the changes weren't pushed to instances of the master, and alignment box information displayed when instancing the edited master wasn't correct.

To edit the shapes and cells of a **Master** object from a program, follow these steps:

Open the **Master** object for editing using *masterObjCopy* = *masterObj*.**Open**. This code fails if there is a drawing window open into *masterObj* or if other programs already have *masterObj* open. If the **Open** method succeeds, *masterObjCopy* is a copy of *masterObj*.

Change any shapes and cells in *masterObjCopy*, not *masterObj*.

Close the **Master** object using *masterObjCopy*.**Close**. The **Close** method fails if *masterObjCopy* isn't a **Master** object that resulted from a prior *masterObj*.**Open** call. Otherwise, the **Close** method merges the changes made in step 2 from *masterObjCopy* back into *masterObj*. It also updates all instances of *masterObj* to reflect the changes and update information cached in *masterObj*. If *masterObj*.**IconUpdate** isn't **visManual** (0), the **Close** method updates the icon shown in the stencil window for *masterObj* to depict an image of *masterObjCopy*.

If you change the shapes and cells of a master directly, as opposed to opening and closing the master as described in the procedure above, the effects listed in step 3 don't occur.

A program that creates a copy of a *masterObj* for editing should both close and release the copy. Microsoft Visual Basic typically releases it automatically. However, with C/C++, you must explicitly release the copy, just as you would for any other object.

OpenDrawWindow method

Example

Opens a new drawing window that displays a page, master, or group.

Version added

4.1

Syntax

objRet = *object*.**OpenDrawWindow**

objRet	A **Window** object that represents the opened window.
object	Required. An expression that returns a **Page**, **Master**, or **Shape** object to display in the drawing window.

Remarks

The **OpenDrawWindow** method opens a new drawing window, even if the page, master, or group is already displayed in a drawing window.

OpenEx method

Opens an existing Microsoft Visio file using extra information passed in an argument.

Version added

4.0

Syntax

objRet = *object*.**OpenEx** (*fileName*, *openFlags*)

objRet	A **Document** object that represents the file that was opened.
object	Required. An expression that returns a **Documents** collection.
fileName	Required **String**. The name of the file.
openFlags	Required **Integer**. Flags that indicate how to open the file.

Remarks

The **OpenEx** method is identical to the **Open** method, except that it provides an extra argument in which the caller can specify how the document opens.

The *openFlags* argument should be a combination of zero or more of the following values.

Constant	Value
visOpenCopy	&H1
visOpenRO	&H2
visOpenDocked	&H4
visOpenDontList	&H8
visOpenMinimized	&H10
visOpenRW	&H20
visOpenMacrosDisabled	&H80

If **visOpenDocked** is specified, the file appears in a docked rather than an MDI window, provided that the file is a stencil file and there is an active drawing window in which to put the docked stencil window.

If **visOpenDontList** is specified, the name of the opened file doesn't appear in the list of recently opened documents on the **File** menu.

If **visOpenMinimized** is specified, the file opens minimized—it is not active. This flag is not supported in versions of Visio earlier than 5.0b.

If **visOpenMacrosDisabled** is specified, the file opens with VBA macros disabled. This flag is not supported in versions earlier than Microsoft Visio 2002.

OpenIconWindow method

Opens an icon window that shows a master's icon.

Version added

4.1

Syntax

objRet = *object*.**OpenIconWindow**

objRet	A **Window** object that represents the opened window.
object	Required. An expression that returns a **Master** object.

Remarks

If the master's icon is already displayed in an icon window, the **OpenIconWindow** method activates that window rather than opening another window.

OpenSheetWindow method

Opens a ShapeSheet window for a **Shape** object.

Version added

4.1

Syntax

objRet = *object*.**OpenSheetWindow**

objRet	A **Window** object that represents the opened window.
object	Required. An expression that returns a **Shape** object.

Remarks

The **OpenSheetWindow** method opens a new ShapeSheet window for the shape even if the information is already displayed in another window.

OpenStencilWindow method

Opens a stencil window that shows the masters in the document.

Version added

4.1

Syntax

objRet = *object*.**OpenStencilWindow**

objRet	A **Window** object that represents the opened window.
object	Required. An expression that returns a **Document** object.

Remarks

If the document's stencil is already displayed in a stencil window, the **OpenStencilWindow** method activates that window rather than opening another window.

ParseLine method

Example

Parses a line of Microsoft Visual Basic code.

Version added

4.5

Syntax

```
object.ParseLine line
```

object	Required. An expression that returns a **Document** object.
line	Required **String**. A string interpreted as Visual Basic for Applications (VBA) code.

Remarks

The **ParseLine** method tells the VBA project of the **Document** object to parse the string. VBA treats the string like it would treat the same string typed into its Immediate window.

The **ParseLine** method creates a Visual Basic project in the document if one

does not exist. The **ParseLine** method raises an exception if the string fails to parse. You can determine whether the string has successfully parsed using the following technique.

```
Public Sub parse(str As String)
    On Error Resume Next
    ThisDocument.ParseLine str
    If Err = 0 Then
        MsgBox "String parsed successfully"
    Else
        MsgBox "Parse not successful"
    End If
End Sub
```

Paste method

Pastes the contents of the Clipboard into an object.

Version added

2002

Syntax

object.**Paste** [*flags*]

object	Required. An expression that returns the **Page**, **Shape**, or **Master** object to paste.
flags	Optional **Variant**. Determines how shapes are translated during the paste operation.

Remarks

The **Paste** method works only with **Shape** objects that are group shapes. Use the **Type** property of a shape to determine whether it is a group.

Possible values for *flags* are declared by the Visio type library in

VisCutCopyPasteCodes, and are described in the following table.

Flag	Value	Description
visCopyPasteNormal	&H0	Default. Shapes are pasted to the center of the active window.
visCopyPasteNoTranslate	&H1	Shapes are pasted to their original coordinate locations.

Setting *flag* to **visCopyPasteNormal** is the equivalent of the behavior in the user interface. You should use the **visCopyPasteNormal** and **visCopyPasteNoTranslate** flags consistently. For example, if you copy using **visCopyPasteNoTranslate**, you should also paste using that value as it is the only way to ensure that shapes are pasted to their original coordinate location.

If you need to control the format of the pasted information and (optionally) establish a link to a source file (for example, a Microsoft Word document), use the **PasteSpecial** method.

Paste method (Characters object)

See also Example

Pastes the text range on the Clipboard into an object.

Version added

2.0

Syntax

object.**Paste**

object	Required. An expression that returns the **Characters** object to paste.

PasteSpecial method

See also Example

Inserts the contents of the Clipboard, allowing you to control the format of the pasted information and (optionally) establish a link to the source file (for example, a Microsoft Word document).

Version added

2002

Syntax

object.**PasteSpecial** (*format* [,*link*][,*displayAsIcon*])

object	Required. An expression that returns a **Master**, **Page**, or **Shape** object.
format	Required **Long**. The internal Clipboard format.
link	Optional **Variant**. **True** to establish a link to the source of the pasted data; otherwise, **False** (the default). Ignored if the source data is not suitable for, or doesn't support, linking.
displayAsIcon	Optional **Variant**. **True** to display the pasted data as an icon; otherwise, **False** (the default).

Remarks

To simply paste the contents of the Clipboard into an object, use the **Paste** method.

The **PasteSpecial** method of a **Shape** object works only with **Shape** objects that

are group shapes. Use the **Type** property of a shape to determine whether it is a group.

The value of the format argument can be any of the following:

A value from **VisPasteSpecialFormat** (see the following table).

Any of the standard Clipboard formats, for example, CF_TEXT. For details, see the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

Any value returned from a call to the **RegisterClipboardFormat** function. For details, see the Microsoft Platform SDK on the MSDN Web site.

Possible values for *format* declared by the Visio type library in **VisPasteSpecialFormat** are described in the following table.

Flag	Value
visPasteText	1
visPasteBitmap	2
visPasteMetafilePicture	3
visPasteOEMText	7
visPasteDeviceIndependentBitmap	8
visPasteEnhancedMetafile	14
visPasteOleObject	65536
visPasteRTF	65537
visPasteHyperlink	65538
visPasteURL	65539

Point method

Returns a point at a position along a curve.

Version added

5.0

Syntax

object.**Point**(*t*, *x*, *y*)

object	Required. An expression that returns a **Curve** object.
t	Required **Double**. The value in the curve's parameter domain to evaluate.
x	Required **Double**. Returns *x* value of curve at *t*.
y	Required **Double**. Returns *y* value of curve at *t*.

Remarks

A **Curve** object describes itself in terms of its parameter domain, which is the range [Start(),End()]. The **Point** method of a **Curve** object returns the *x*,*y*

coordinates at position *t*, which is any position along the curve's path. The **Point** method can be used to extrapolate the curve's path outside of [Start(),End()].

PointAndDerivatives method

Returns a point and derivatives at a position along a curve's path.

Version added

5.0

Syntax

object.**PointAndDerivatives**(*t*, *n*, *x*, *y*, *dx*, *dy*, *ddx*, *ddy*)

object	Required. An expression that returns a **Curve** object.
t	Required **Double**. The value in the curve's parameter domain to evaluate.
n	Required **Integer**. 0: get point; 1: point and 1st derivative; 2: point plus first and second derivative.
x	Required **Double**. Returns *x* value of curve at *t*.
y	Required **Double**. Returns *y* value of curve at *t*.
dx	Required **Double**. Returns first derivative (*dx/dt*) at *t* if *n* > 0.
dy	Required **Double**. Returns first derivative (*dx/dt*) at *t* if *n* > 0.
ddx	Required **Double**. Returns second derivative (*ddx/dt*) at *t* if *n* >

	1.
ddy	Required **Double**. Returns second derivative (*ddy/dt*) at *t* if *n* > 1.

Remarks

Use the **PointAndDerivatives** method of the **Curve** object to obtain the coordinates of a point within the curve's parameter domain and its first and second derivatives.

A **Curve** object describes itself in terms of its parameter domain which is the range [Start(),End()]. The **PointAndDerivatives** method can be used to extrapolate the curve's path outside [Start(),End()].

Print method

See also Example

Prints the contents of an object to the default printer.

Version added

2.0

Syntax

object.**Print**

object	Required. An expression that returns a **Page** or **Document** object to print.

Remarks

For a **Document** object, this method prints all of the document's pages. Background pages are printed on the same sheet of paper as the foreground pages to which they are assigned.

For a **Page** object, this method prints the page and its background page (if any) on the same sheet of paper.

If you're using Microsoft Visual Basic for Applications or Visual Basic, you must assign the method result to a dummy variable and you must apply the method to a variable of type **Object**, not type **Visio.Document** or **Visio.Page**. For example, to print a document:

```
Dim docObj As Visio.Document
Dim docObjTemp as Object
Dim dummy As String
Set docObj = ThisDocument
Set docObjTemp = docObj
dummy = docObjTemp.Print
```

PrintTile method

Example

Prints a single tile of a drawing page.

Version added

2002

Syntax

object.**PrintTile** *nTile*

object	Required. An expression that returns a **Page** object to print.
nTile	Required **Long**. The number of tiles.

Remarks

Use the **PrintTile** method to print a single tile of a drawing that spans multiple physical printer pages.

This method is the equivalent of clicking the **Print** toolbar button when you are previewing a single tile in **Print Preview** mode.

PurgeUndo method

Empties the Microsoft Visio queue of undo actions.

Version added

5.0

Syntax

object.**PurgeUndo**

object	Required. An expression that returns an **Application** object.

Remarks

After calling the **PurgeUndo** method, no operation performed before the call can be reversed.

QueueMarkerEvent method

Queues a marker event that fires after all other queued events.

Version added

5.0

Syntax

intRet = object.**QueueMarkerEvent**(*contextString*)

intRet	**Long**. The sequence number of the event that fires after all other queued events.
object	Required. An expression that returns an **Application** object.
contextString	Required **String**. An arbitrary string that is passed with the event that fires.

Remarks

The **QueueMarkerEvent** method works in conjunction with the **MarkerEvent** event to allow an Automation client to queue an event to itself. The **QueueMarkerEvent** method causes the application to fire a **MarkerEvent**

event after it has fired all of the events in its event queue.

The **QueueMarkerEvent** method returns the sequence number of the **MarkerEvent** event to fire, and the string passed to the **QueueMarkerEvent** method (legally empty) is passed to the **MarkerEvent** event handler.

A client program can use either the sequence number or the string to correlate **QueueMarkerEvent** calls with **MarkerEvent** events. In this way, the client is able to distinguish events it caused and events it did not cause.

Quit method

Example

Closes the indicated instance of Microsoft Visio.

Version added

2.0

Syntax

```
object.Quit
```

object	Required. An expression that returns an **Application** object.

Remarks

If the **Quit** method is invoked when a document with unsaved changes is open, a dialog box appears asking if you want to save the document. To quit the application without saving and seeing the dialog box, set the **Saved** property of the **Document** object representing the document to **True** immediately before quitting. Set the **Saved** property to **True** only if you are sure you want to close the document without saving changes.

Redo method

Reverses the most recent undo unit.

Version added

2.0

Syntax

object.**Redo**

object	Required. An expression that returns an **Application** object.

Remarks

To reverse the effect of the **Undo** method, use the **Redo** method. For example, if you clear an item and restore it with the **Undo** method, use the **Redo** method to clear the item again.

You cannot invoke the **Redo** method from code that is executing inside the scope of an open undo unit. Code is in the scope of an open undo unit if it is one of the following:

A macro or add-on invoked by the Visio user interface.

In an event handler responding to a Visio event other than the **VisioIsIdle** event.

In a user-created undo scope. If you call the **Redo** method from code inside the scope of an open undo unit, it will raise an exception.

The **Redo** method also raises an exception if the Visio instance is presently performing an undo or redo. To determine whether the Visio instance is undoing or redoing use the **IsUndoingOrRedoing** property.

You can call the **Redo** method from the **VisioIsIdle** event handler because the **VisioIsIdle** event can only fire when the **IsUndoingOrRedoing** property is **False**. You can also call the **Redo** method from code not invoked by the Visio instance, for example, code invoked from the Visual Basic Editor or from an external program.

Remove method

See also Example

Removes a shape from a layer.

Version added

4.0

Syntax

object.**Remove** *shapeObj*, *fPreserveMembers*

object	Required. An expression that returns a **Layer** object.
shapeObj	Required. An expression that returns the **Shape** object to remove.
fPreserveMembers	Required **Integer**. Flag that indicates whether to remove members of a group.

Remarks

If the shape is a group and *fPreserveMembers* is non-zero, member shapes of the group are unaffected. If *fPreserveMembers* is zero (0), the group's member shapes are also removed from the layer.

Removing a shape from a layer does not delete the shape.

RemoveFromGroup method

Example

Removes selected shapes from a group.

Version added

2.0

Syntax

object.**RemoveFromGroup**

object	Required. An expression that returns a **Selection** object.

RenameCurrentScope method

Example

Renames the top-level open undo scope.

Version added

2002

Syntax

object.**RenameCurrentScope** *bstrScopeName*

object	Required. An expression that returns an **Application** object that contains the undo scope.
bstrScopeName	Required **String**. The new name of the undo scope.

Remarks

The new name assigned to the undo scope appears on the **Undo** menu as the item name. If there is no open undo scope, the **RenameCurrentScope** method raises an exception.

ResizeToFitContents method

See also Example

Resizes the page, or the master's page, to fit tightly around the shapes or master that are on it.

Version added

2002

Syntax

object.**ResizeToFitContents**

object	Required. An expression that returns a **Master** or **Page** object.

Remarks

After the page is resized, the page height and width, and the PinX and PinY values of the shapes or master are typically changed.

The **ResizeToFitContents** method is the equivalent of clicking **Size to fit drawing contents** on the **Page Size** tab in the **Page Setup** dialog box (on the **File** menu, click **Page Setup**).

ReverseEnds method

Example

Reverses an object by flipping it both horizontally and vertically.

Version added

2.0

Syntax

object.**ReverseEnds**

object	Required. An expression that returns the **Shape** or **Selection** object to reverse.

Rotate90 method

Example

Rotates an object 90 degrees counterclockwise.

Version added

2.0

Syntax

object.**Rotate90**

object	Required. An expression that returns the **Shape** or **Selection** object to rotate.

Run method

See also Example

Runs the add-on represented by an **Addon** object.

Version added

4.0

Syntax

object.**Run** *argString*

object	Required. An expression that returns an **Addon** object.
argString	Required **String**. The argument string to pass to the add-on.

Remarks

If the add-on is implemented by an EXE file, the arguments are passed in the command line string. If the add-on is implemented by a VSL file, the arguments are passed in a field of the argument structure that accompanies the run message sent to the VSL's **VisioLibMain** procedure.

Save method

Example

Saves a document.

Version added

2.0

Syntax

object.**Save**

object	Required. An expression that returns a **Document** object.

Remarks

Use the **SaveAs** method to save and name a new document. Until a document has been saved, the **Save** method generates an error.

SaveAs method

Saves a document with a file name.

Version added

2.0

Syntax

object.**SaveAs** *fileName*

object	Required. An expression that returns a **Document** object.
fileName	Required **String**. The file name for the document.

Remarks

The **SaveAs** method can accept drive names using the universal naming convention (UNC), for example, \\corporation\marketing.

Beginning with Visio 2002, you can save your drawing as an XML drawing (.vdx), an XML stencil (.vsx), or an XML template (.vtx).

SaveAsEx method

Saves a document with a file name using extra information passed in an argument.

Version added

4.0

Syntax

object.**SaveAsEx** *fileName*, *saveFlags*

object	Required. An expression that returns a **Document** object.
fileName	Required **String**. The file name for the document.
saveFlags	Required **Integer**. How to save the file.

Remarks

The **SaveAsEx** method is identical to the **SaveAs** method, except that it provides an extra argument in which the caller can specify how the document is to be saved.

The *saveFlags* argument should be a combination of the following values.

Constant	Value	Description
visSaveAsRO	&H1	The document is saved as read-only.
visSaveAsWS	&H2	The current workspace is saved with the file.
visSaveAsListInMRU	&H4	The document is included in the Most Recently Used (MRU) list. By default, **Save** and **SaveAs** do not place the document into the MRU list.

SaveToFile method

Saves the user interface represented by a **UIObject** object in a file.

Version added

4.0

Syntax

object.**SaveToFile** *fileName*

object	Required. An expression that returns the **UIObject** object to save to the file.
fileName	Required **String**. The name of the file in which to save the **UIObject** object.

Remarks

The file can be loaded into the application by using the **LoadFromFile** method of a **UIObject** object.

Note Beginning with Visio 2000, you can customize the user interface by right-

clicking the toolbar, and then clicking **Customize** on the shortcut menu. Changes you make to the interface persist when you close the application—they are stored with the **Application** object and in a file named Custom.vsu, which is stored as Application Data in the current user's User Profile.

SaveWorkspaceAs method

See also Example Applies to

Beginning with Microsoft Visio 2002, this method is obsolete.

Remarks

In versions earlier than 2002, **SaveWorkSpaceAs** saved the workspace into a VSW file.

Scroll method

Example

Scrolls the contents of a window vertically, horizontally, or both.

Version added

2000

Syntax

object.**Scroll** (*longFlagsX*, *longFlagsY*)

object	Required. An expression that returns a **Window** object.
longFlagsX	Required **Long**. Indicates how to scroll horizontally.
longFlagsY	Required **Long**. Indicates how to scroll vertically.

Remarks

Constants representing ways to scroll are declared by the Visio type library in **VisWindowScrollX** and **VisWindowScrollY**.

- Values of *longFlagsX*

- Values of *longFlagsY*

If the **Window** object is not a built-in MDI or built-in docked stencil type, then the **Scroll** method raises an exception.

ScrollViewTo method

Example

Scrolls a window to a particular page coordinate.

Version added

2000

Syntax

object.**ScrollViewTo** (*doubleX*, *doubleY*)

object	Required. An expression that returns a **Window** object.
doubleX	Required **Double**. The *x*-coordinate to which to scroll.
doubleY	Required **Double**. The *y*-coordinate to which to scroll.

Remarks

The **ScrollViewTo** method scrolls to the *doubleX* and *doubleY* coordinates.

If the value of the **Window** object's **Type** property is not **visDrawing**, then the method raises an exception.

Select method

Selects or deselects an object.

Version added

2.0

Syntax

object.**Select** *addObj*, *selectType*

object	Required. An expression that returns a **Window** or **Selection** object that contains the shapes.
addObj	Required. An expression that returns a **Shape** object to select or deselect.
selectType	Required **Integer**. The type of selection to make.

Remarks

When used with the **Window** object, the **Select** method will affect the selection in the Visio window. The **Selection** object, however, is independent of the selection in the window. Therefore, using the **Select** method with a **Selection**

object only affects the state of the object in memory—the Visio window is unaffected.

The following constants declared by the Visio type library show valid values for selection types.

Constant	Value
visDeselect	1
visSelect	2
visSubSelect	3
visSelectAll	4
visDeselectAll	256

You can combine **visDeselectAll** with **visSelect** and **visSubSelect** to deselect all shapes prior to selecting or subselecting other shapes.

If the object being operated on is a **Selection** object, and the **Select** method selects a **Shape** object whose **ContainingShape** property is different than the **ContainingShape** property of the **Selection** object, then the **Select** method deselects everything, even if the selection type value doesn't specify deselection.

SelectAll method

Selects all possible shapes in a window or selection.

Version added

2.0

Syntax

object.**SelectAll**

object	Required. An expression that returns a **Window** or **Selection** object that contains the shapes.

Remarks

All shapes that can be selected are immediate children of the selection's containing shape.

SendBackward method

Moves a shape or selected shapes back one position in the *z*-order.

Version added

2.0

Syntax

object.**SendBackward**

object	Required. An expression that returns a **Shape** or **Selection** object to send backward.

SendToBack method

Moves the shape or selected shapes to the back of the *z*-order.

Version added

2.0

Syntax

object.**SendToBack**

object	Required. An expression that returns a **Shape** or **Selection** object to send to the back.

SetBegin method

Example

Moves the begin point of a one-dimensional (1-D) shape to the coordinates represented by *x* and *y*.

Version added

2.0

Syntax

object.**SetBegin** *x*, *y*

object	Required. An expression that returns a **Shape** object.
x	Required **Double**. The new *x*-coordinate of the begin point.
y	Required **Double**. The new *y*-coordinate of the begin point.

Remarks

The **SetBegin** method only applies to 1-D shapes. If the indicated shape is a 2-D shape, an error is generated.

The coordinates represented by the *x* and *y* arguments are parent coordinates,

measured from the origin of the shape's parent (the page or group that contains the shape).

SetCenter method

Example

Moves a shape so that its pin is positioned at the coordinates represented by *x* and *y*.

Version added

2.0

Syntax

object.**SetCenter** *x*, *y*

object	Required. An expression that returns a **Shape** object.
x	Required **Double**. The new *x*-coordinate of the center of rotation (PinX).
y	Required **Double**. The new *y*-coordinate of the center of rotation (PinY).

Remarks

The coordinates represented by the *x* and *y* arguments are parent coordinates, measured from the origin of the shape's parent (the page or group that contains

the shape).

The **SetCenter** method only moves the point, in parent coordinates, about which the shape rotates. It does not change the point, in local coordinates, about which the shape rotates.

SetCustomMenus method

Replaces the current built-in or custom menus of an application or document.

Version added

4.0

Syntax

object.**SetCustomMenus** *UIObject*

object	Required. An expression that returns an **Application** or **Document** object to receive the custom menus.
UIObject	Required. An expression that returns a **UIObject** object that represents the new custom menus.

Remarks

If the **UIObject** object was created in a separate process by using the **CreateObject** procedure instead of getting the appropriate property of an **Application** or **Document** object, the **SetCustomMenus** method returns an error.

SetCustomToolbars method

Replaces the current built-in or custom toolbars of an application or document.

Version added

4.0

Syntax

object.**SetCustomToolbars** *UIObject*

object	Required. An expression that returns an **Application** or **Document** object to receive the custom toolbars.
UIObject	Required. An expression that returns a **UIObject** object that represents the new custom toolbars.

Remarks

If the **UIObject** object was created in a separate process by using the **CreateObject** procedure instead of getting the appropriate property of an **Application** or **Document** object, the **SetCustomToolbars** method returns an error.

SetEnd method

Example

Moves the end point of a one-dimensional (1-D) shape to the coordinates represented by *x* and *y*.

Version added

2.0

Syntax

object.**SetEnd** *x*, *y*

object	Required. An expression that returns a **Shape** object.
x	Required **Double**. The new *x*-coordinate of the end point.
y	Required **Double**. The new *y*-coordinate of the end point.

Remarks

The **SetEnd** method applies only to 1-D shapes. If the indicated shape is a 2-D shape, an error is returned.

The coordinates represented by the *x* and *y* arguments are parent coordinates,

measured from the origin of the shape's parent (the page or group that contains the shape).

SetFilterCommands method

Example

Specifies an array of command ranges and a **True** or **False** value indicating how to filter events for each command range.

Version added

2002

Syntax

object.**SetFilterCommands** *commands*

object	Required. An expression that returns an **Event** object.
commands	Required **Long**. An array of command ranges and a **True** or **False** value specifying how to filter events for each command range.

Remarks

When an **Event** object created with the **AddAdvise** method is added to the **EventList** collection of a source object, the default behavior is that all occurrences of that event are passed to the event sink. The **SetFilterCommands**

method provides a way of ignoring selected events based on command ID.

The *command*s argument passed to **SetFilterCommands** is an array defined in the following way.

The number of elements in *commands* is a multiple of 3:

The first element contains the beginning command ID of the range (any member of **VisUICmds**).

The second element contains the end command ID of the range (any member of **VisUICmds**).

The third element contains a **True** or **False** value, which indicates whether you are listening to events for that command range (**True** to listen to events; **False** to exclude events).

For an event to successfully pass through a command filter, it must satisfy the following criteria:

It must have a valid command ID.

If all filters are **True**, the event must match at least one filter.

If all filters are **False**, the event must not match any filter.

If the filters are a mixture of **True** and **False**, the event must match at least one **True** filter and not match any **False** filters.

If there are no **True** ranges in the array, events are considered **True**.

For example, to set up an array that blocks out a single command, use the following:

```
Dim cmdArray (1 * 3) As Long
'Ignore the layout command
cmdArray(1) = visCmdLayoutDynamic
cmdArray(2) = visCmdLayoutDynamic
cmdArray(3) = False
```

Or, to set up an array that listens only to the **Send to Back** command:

```
Dim cmdArray (3 * 3) As Long
' Pay attention to the Send To Back command
cmdArray(1) = visCmdObjectSendToBack
cmdArray(2) = visCmdObjectSendToBack
cmdArray(3) = True
'Ignore any command IDs before the Send To Back comm
commands(4) = visCmdCMDFIRST
commands(5) = visCmdObjectSendToBack - 1
cmdArray(6) = False
'Ignore any command IDs after the Send To Back commar
commands(4) = visCmdObjectSendToBack + 1
commands(5) = visCmdCMDLAST
commands(6) = False
```

SetFilterObjects method

Example

Specifies an array of object types and a **True** or **False** value indicating how to filter events for each object.

Version added

2002

Syntax

object.**SetFilterObjects** *objects*

object	Required. An expression that returns an **Event** object.
objects	Required **Long**. An array of objects types and a **True** or **False** value specifying how to filter events for each object type.

Remarks

When an **Event** object created with the **AddAdvise** method is added to the **EventList** collection of a source object, the default behavior is that all occurrences of that event are passed to the event sink. The **SetFilterObjects** method provides a way to ignore selected events based on object type.

The *objects* argument passed to **SetFilterObjects** is an array defined in the following manner.

The number of elements in the array is a multiple of 2:

The first element contains an object type (one of **visTypePage**, **visTypeGroup**, **visTypeShape**, **visTypeForeignObject**, **visTypeGuide**, or **visTypeDoc**).

The second element contains a **True** or **False** value indicating whether you are listening to events for that object (**True** to listen to an object's events; **False** to exclude an object's events).

For an event to successfully pass through an object event filter, it must satisfy the following criteria:

It must be a valid object type.

If all filters are **True**, the event must match at least one filter.

If all filters are **False**, the event must not match any filter.

If the filters are a mixture of **True** and **False**, the event must match at least one **True** filter and not match any **False** filters.

If there are no **True** ranges defined in the array, events are considered **True**.

For example, if you want to listen only to events sourced by a shape or guide, you can pass an array like the following:

```
Dim objArray (2 * 2) As Long
objArray(1) = visTypeShape
objArray(2) = True
objArray(3) = visTypeGuide
objArray(4) = True
```

SetFilterSRC method

Example

Specifies an array of cell ranges and a **True** or **False** value indicating how to filter events for each cell range.

Version added

2002

Syntax

object.**SetFilterSRC** *SRCStream*

object	Required. An expression that returns an **Event** object.
SRCStream	Required **Integer**. An array of cell ranges and a **True** or **False** value specifying how to filter events for each range.

Remarks

When an **Event** object created with the **AddAdvise** method is added to the **EventList** collection of a source object, the default behavior is that all occurrences of that event are passed to the event sink. The **SetFilterSRC** method provides a way to ignore selected events based on a range of cells.

The *SRCStream* argument passed to **SetFilterCommands** is an array defined in the following manner:

The number of elements in the array is a multiple of 7:

The first three elements describe the section, row, and cell of the beginning cell of the range.

The next three elements describe the section, row, and cell of the end cell of the range.

The last element contains a **True** or **False** value indicating how to filter events for the cell range (**True** to listen to events for a range of cells; **False** to exclude events for a range of cells).

For an event to successfully pass through a cell range filter, it must satisfy the following criteria:

It must be a valid section, row, cell reference.

If all filters are **True**, the event must match at least one filter.

If all filters are **False**, the event must not match any filter.

If the filters are a mixture of **True** and **False**, the event must match at least one **True** filter and not match any **False** filters.

If there are no **True** ranges defined in the array, events are considered **True**.

For example, if you want to listen for any changes in the Value cell of the second row in the Custom Property section, use the following:

```
Dim srcArray (1 * 7) As Long
srcArray(1) = visSectionProp
srcArray(2) = visRowProp + 1
srcArray(3) = visCustPropsValue
srcArray(4) = visSectionProp
srcArray(5) = visRowProp + 1
```

srcArray(6) = visCustPropsValue
srcArray(7) = True

SetFormulas method

Sets the formulas of one or more cells.

Version added

4.5

Syntax

intRet object.**SetFormulas** *SID_SRCStream*, *formulas*, *fla*

intRet	**Integer**. Number of *SID_SRCStream* entries which processed successfully.
object	Required. An expression that returns a **Page**, **Master**, **Shape**, o **Style** object whose cells are to be modified.
SID_SRCStream	Required **Integer**. Stream identifying cells to be modified.
formulas	Required **Variant**. Formulas to be assigned to identified cells.
flags	Required **Integer**. Flags that influence the behavior of **SetFormulas**.

Remarks

The **SetFormulas** method behaves like the **Formula** property, except you can use it to set the formulas of many cells at once, rather than one cell at a time.

For **Shape** or **Style** objects, you can use the **SetFormulas** method to set results of any set of cells.

For **Page** or **Master** objects, you can use the **SetFormulas** method to set results of any set of cells in any set of shapes of the page or master.

In both of these cases, you tell the **SetFormulas** method which cells you want to set by passing an array of integers in *SID_SRCStream. SID_SRCStream* is a one-dimensional array of 2-byte integers.

For **Shape** and **Style** objects, *SID_SRCStream* should be a one-dimensional array of 3*n* 2-byte integers for *n* >= 1. The **SetFormulas** method interprets the stream as:

{ sectionIdx, rowIdx, cellIdx }n

where *sectionIdx* is the section index of the desired cell, *rowIdx* is its row index, and *cellIdx* is its cell index.

For **Page** and **Master** objects *SID_SRCStream* should be a one-dimensional array of 4*n* 2-byte integers for *n* >= 1. The **SetFormulas** method interprets the stream as:

{ sheetID, sectionIdx, rowIdx, cellIdx }n

where *sheetID* is the **ID** property of the **Shape** object on the page or master whose cell result is to be modified.

If the *sheetID* in an entry is **visInvalShapeID** (-1) or if the bottom byte of *sectionIdx* is **visSectionInval** (255), then the entry is ignored by the **SetResults** method. The motivation for this is that the same *SID_SRCStream* array can be used on several calls to **SetFormulas**, **GetFormulas**, and similar methods with the caller only needing to make minor changes to the stream between calls.

The *formulas* argument should be a one-dimensional array of 1 <= *m* variants. Each **Variant** should be a **String**, a reference to a **String**, or **Empty**. If

formulas(*i*) is empty, then the *i*'th cell will be set to the formula in formulas(*j*), where *j* is the index of the most recent prior entry which is not empty. If there is no prior entry that is not empty, the corresponding cell is not altered. If fewer formulas than cells are specified (*m* < *n*), then the *i*'th cell, *i* > *m*, will be set to the same formula as was chosen to set the *m*'th cell to. Thus to set many cells to the same formula, you need only pass one copy of the formula.

The *flags* argument should be a bit mask of the following values.

Constant	Value	Description
visSetBlastGuards	&H2	Override present cell values even if they're guarded.
visSetTestCircular	&H4	Test for establishment of circular cell references.
visSetUniversalSyntax	&H8	Formulas are in universal syntax.

The value returned by the **SetFormulas** method is the number of entries in *SID_SRCStream* that were successfully processed. If *i* < *n* entries process correctly, but an error occurs on the *i* + 1st entry, then the **SetFormulas** method raises an exception and returns *i*. Otherwise, *n* is returned.

SetResults method

Sets the results or formulas of one or more cells.

Version added

4.5

Syntax

intRet object.**SetResults** *SID_SRCStream*, *units*, *results*, *fl*

intRet	**Integer**. Number of *SID_SRCStream* entries which processed successfully.
object	Required. An expression that returns a **Page**, **Master**, **Shape**, o **Style** whose cells are to be modified.
SID_SRCStream	Required **Integer**. An array identifying cells to be modified.
units	Required **Variant**. Measurement units to be attributed to entries results array.
results	Required **Variant**. Results or formulas to be assigned to identif cells.
flags	Required **Integer**. Flags that influence the behavior of **SetResu**

Remarks

The **SetResults** method is like the **Result** method of a **Cell** object, except that it can be used to set the results (values) of many cells at once, rather than one cell at a time.

For **Shape** or **Style** objects, you can use the **SetResults** method to set results of any set of cells.

For **Page** or **Master** objects, you can use the **SetResults** method to set results of any set of cells in any set of shapes of the page or master.

In both of these cases, you tell the **SetResults** method which cells you want to set by passing an array of integers in *SID_SRCStream. SID_SRCStream* is a one-dimensional array of 2-byte integers.

For **Shape** and **Style** objects *SID_SRCStream* should be a one-dimensional array of 3*n* 2-byte integers for *n* >= 1. The **SetResults** method interprets the stream as:

{ sectionIdx, rowIdx, cellIdx }n

where *sectionIdx* is the section index of the desired cell, *rowIdx* is its row index, and *cellIdx* is its cell index.

For **Page** and **Master** objects *SID_SRCStream* should be a one-dimensional array of 4*n* 2-byte integers for *n* >= 1. The **SetResults** method interprets the stream as:

{ sheetID, sectionIdx, rowIdx, cellIdx }n

where *sheetID* is the **ID** property of the **Shape** object on the page or master whose cell result is to be modified.

If the *sheetID* in an entry is **visInvalShapeID** (-1) or if the bottom byte of *sectionIdx* is **visSectionInval** (255), then the entry is ignored by the **SetResults** method. The motivation for this is that the same *SID_SRCStream* array can be used on several calls to **SetResults**, **GetResults**, and similar methods with the caller only needing to make minor changes to the stream between calls.

The *units* array controls what measurement units individual entries in results are in. Each entry in the array can be a string such as "inches", "inch", "in.", or "i". Strings may be used for all supported Visio units such as centimeters, meters, miles, and so on. You can also indicate desired units with integer constants (**visCentimeters**, **visInches**, etc.) declared by the Visio type library in **VisUnitCodes**. For a list of constants used for units of measure, see About units of measure. Note that the values specified in the *units* array have no effect if **visSetFormulas** is set in *flags*.

If not empty, we expect *units* to be a one-dimensional array of 1 <= *u* variants. Each entry can be a string or integer code, or empty (nothing). If the *i*'th entry is empty, then the *i*'th entry in results is in the units designated by *units*(*j*), where *j* is the most recent prior entry that is not empty. Thus if you want all entries in *results* to be interpreted in the same units, then you need only pass a *units* array with one entry. If there is no prior entry that is not empty, or if no *units* array is supplied, then **visNumber** (0x20) will be used. This causes the application to default to internal units (like the **ResultIU** property of a **Cell** object).

The *results* argument should be a one-dimensional array of 1 <= *m* variants. A result can be passed as **Double**, **Integer**, **String**, or a reference to a **String**. Strings are accepted only if **visSetFormulas** is set in *flags*, in which case strings are interpreted as formulas. If *results(i*) is empty, then the *i*'th cell will be set to the value in *results*(*j*), where *j* is the index of the most recent prior entry which is not empty. If there is no prior entry that is not empty, the corresponding cell is not altered. If fewer results than cells are specified (*m* < *n*), then the *i*'th cell, *i* < *m*, will be set to the same value as was chosen to set the *m*'th cell to. Thus to set many cells to the same value, you need only pass one copy of the value.

The *flags* argument should be a bit mask of the following values.

Constant	Value	Description
visSetFormulas	&H1	Treat strings in results as formulas.
visSetBlastGuards	&H2	Override present cell values even if they're guarded.
visSetTestCircular	&H4	Test for establishment of circular cell references.
visSetUniversalSyntax	&H8	Formulas are in universal syntax

The value returned by the **SetResults** method is the number of entries in *SID_SRCStream* that were successfully processed. If $i < n$ entries process correctly, but an error occurs on the i + 1st entry, then the **SetResults** method raises an exception and returns i. Otherwise, n is returned.

SetViewRect method

Example

Sets the page coordinates of a window's borders by adjusting the zoom level and center scroll position.

Version added

2000

Syntax

object.**SetViewRect** *dLeft*, *dTop*, *dWidth*, *dHeight*

object	Required. An expression that returns a **Window** object.
dLeft	Required **Double**. The page coordinate of the left side of the window.
dTop	Required **Double**. The page coordinate of the top of the window.
dWidth	Required **Double**. The distance in page units from the left side to the right side of the window.
dHeight	Required **Double**. The distance in page units from the top to the bottom of the window.

Remarks

If the **Window** object is not a **visDrawing** type, then the **SetViewRect** method raises an exception.

SetWindowRect method

Example

Sets the size and position of the client area of a window.

Version added

2000

Syntax

object.**SetWindowRect** *nLeft*, *nTop*, *nWidth*, *nHeight*

object	Required. An expression that returns a **Window** object.
nLeft	Required **Long**. The coordinate of the left side of the window.
nTop	Required **Long**. The coordinate of the top of the window.
nWidth	Required **Long**. The distance in pixels from the left side to the right side of the window.
nHeight	Required **Long**. The distance in pixels from the top to the bottom of the window.

Remarks

The **SetWindowRect** method sets the size and position of the client area of the window with respect to the window that owns the **Windows** collection to which it belongs. For the **Windows** collection of an **Application** object, the "with respect to" window is the MDICLIENT window of the Visio main window. For the **Windows** collection of a **Window** object, the "with respect to" window is the client area of the drawing window.

Subtract method

Example

Subtracts the areas that overlap the selected shape.

Version added

4.0

Syntax

object.**Subtract**

object	Required. An expression that returns a **Selection** object that contains the shapes to subtract.

Remarks

The **Subtract** method is equivalent to clicking **Subtract** on the **Operations** submenu on the **Shape** menu in Visio. The first selected shape is the one that will have the other selected shapes subtracted from it. The other shapes will be deleted and no shapes are selected when the operation is complete.

SwapEnds method

See also Example

Swaps the begin and end points of a one-dimensional (1-D) shape.

Version added

2002

Syntax

object.SwapEnds

object	Required. An expression that returns a **Selection** or **Shape** object.

Remarks

The type of glue associated with the endpoints is also swapped. For example, if the begin point of a 1-D shape is glued to object A and the end point of the 1-D shape is not glued, then, after invoking the **SwapEnds** method, the end point will be glued to object A and the begin point will not be glued.

TransformXYFrom method

Example

Transforms a point expressed in the local coordinate system of one **Shape** object from an equivalent point expressed in the local coordinate system of another **Shape** object.

Version added

2000

Syntax

object.**TransformXYFrom** *OtherShape*, *x*, *y*, *xprime*, *ypr*

object	Required. An expression that returns a **Shape** object whose loca coordinate system you are transforming the point to.
OtherShape	Required. An expression that returns a **Shape** object whose loca coordinate system you are transforming the point from.
x	Required **Double**; *x*-coordinate corresponding to *x* in the *OtherShape* coordinate system.
y	Required **Double**; *y*-coordinate corresponding to *y* in the *OtherShape* coordinate system.

xprime	Required **Double**; *x*-coordinate in coordinate system of *object.*
yprime	Required **Double**; *y*-coordinate in coordinate system of *object.*

Remarks

The points *x*, *y*, *xprime*, and *yprime* are all treated as internal drawing units.

An exception is raised if *object* is not a **Shape** object of a **Page** or **Master** object, or if *OtherShape* is not in the same **Page** or **Master** object as *object*.

TransformXYTo method

Example

Transforms a point expressed in the local coordinate system of one **Shape** object to an equivalent point expressed in the local coordinate system of another **Shape** object.

Version added

2000

Syntax

object.**TransformXYTo** *OtherShape*, *x*, *y*, *xprime*, *yprime*

object	Required. An expression that returns a **Shape** object whose local coordinate system you are transforming the point from.
OtherShape	Required. An expression that returns a **Shape** object whose local coordinate system you are transforming the point to.
x	Required **Double**; *x*-coordinate in coordinate system of *object.*
y	Required **Double**; *y*-coordinate in coordinate system of *object.*
xprime	Required **Double**; *x*-coordinate corresponding to *x* in the *OtherShape* coordinate system.

yprime	Required **Double**; *y*-coordinate corresponding to *y* in the *OtherShape* coordinate system.

Remarks

The points *x*, *y*, *xprime* and *yprime* are all treated as internal drawing units.

An exception is raised if *object* is not a **Shape** object of a **Page** or **Master** object, or if *OtherShape* is not in the same **Page** or **Master** object as *object*.

Trigger method (Cell object)

Example

Evaluates the formula of a cell.

Version added

4.0

Syntax

object.**Trigger**

object	Required. An expression that returns a **Cell** object.

Remarks

Triggering a cell simply evaluates the formula of that cell. If the formula contains other actions such as running an add-on, those actions occur.

Trigger method (Event object)

See also Example

Causes an event's action to be performed.

Version added

4.0

Syntax

object.**Trigger** *contextString*

object	Required. An expression that returns an **Event** object.
contextString	The string to send to the target of the event.

Remarks

Triggering an event causes the action associated with the event to be performed. The specified context string is passed to the target of the action:

If the action is to run an add-on (**visEvtCodeRunAddon**), the string is passed in the command line string sent to the add-on.

If the action is to send a notification to the calling program (**visEvtCodeAdvise**), the string is passed in the *moreInfo* parameter of the notification.

Trim method

Example

Trims selected shapes into smaller shapes.

Version added

4.1

Syntax

object.**Trim**

object	Required. An expression that returns a **Selection** object that contains the shapes to trim.

Remarks

The **Trim** method is equivalent to clicking **Trim** on the **Operations** submenu on the **Shape** menu in Visio.

The new shapes inherit the formatting of the first selected shape, have no text, and are the topmost shapes in their container—the *n*th shape, *n*th – 1 shape, *n*th – 2 shape, and so forth in the **Shapes** collection of their containing shape, where *n*

= count. The original shapes are deleted and no shapes are selected when the operation is complete.

The **Trim** method is similar to the **Fragment** method but differs in the following ways:

Shapes produced by the **Trim** method coincide with the distinct *paths* of the selected shapes, also taking overlap into account.

Shapes produced by the **Fragment** method coincide with the distinct *regions* of the selected shapes, taking overlap into account.

Undo method

Reverses the most recent undo unit, if the undo unit can be reversed.

Version added

2.0

Syntax

```
object.Undo
```

object	Required. An expression that returns an **Application** object.

Remarks

Use the **Undo** method to reverse actions one undo unit at a time.

The number of times that code can call the **Undo** method depends on whether or not the code is executing in the scope of an open undo unit. Code runs in the scope of an open undo unit if it is:

A macro or add-on invoked by the Visio user interface.

In an event handler responding to a Visio event other than the **VisioIsIdle** event.

In a user-created undo scope.

If *code is not executing in the scope of an open undo unit*, it can call the **Undo** method for each undo unit presently on the Visio undo stack. The maximum number of units on the undo stack is set in the **Options** dialog box on the **General** tab (20 is the default). If the number of calls to the **Undo** method exceeds the number of undo units on the stack, no action is taken and the **Undo** method raises no exception.

If *code is executing in the scope of an open undo unit*, it can call the **Undo** method once for each operation in the open undo unit. If there are additional calls to the **Undo** method, it will raise an exception and take no action. For example, if code in a macro performs two operations, it can call the **Undo** method twice. If the macro calls the **Undo** method a third time, the **Undo** method will raise an exception.

Code that calls the **Undo** method from within the scope of an undo unit cannot call the **Redo** method to reverse the action. The **Redo** method can only be called when there are no open undo units.

The **Undo** method also raises an exception if the Visio instance is presently performing an undo or redo. To determine whether the Visio instance is undoing or redoing, use the **IsUndoingOrRedoing** property.

You can call the **Undo** method from the **VisioIsIdle** event handler because the **VisioIsIdle** event can only fire when the **IsUndoingOrRedoing** property is **False**. You can also call the **Undo** method from code not invoked by the Visio instance, for example, code invoked from the Visual Basic Editor or from an external program.

You can undo most actions, but not all. Use the **Redo** method to reverse the effect of the **Undo** method.

Ungroup method

Ungroups a group.

Version added

2.0

Syntax

object.**Ungroup**

object	Required. An expression that returns a **Shape** or **Selection** object to ungroup.

Union method

Example

Creates a new shape from the perimeter of selected shapes.

Version added

2.0

Syntax

object.**Union**

object	Required. An expression that returns a **Selection** object that contains the shapes to unite.

Remarks

The **Union** method is equivalent to clicking **Union** on the **Operations** submenu on the **Shape** menu in Visio. The produced shape will be the topmost shape in its containing shape and will inherit the text and formatting of the first selected shape.

The original shapes are deleted and no shapes are selected when the operation is

complete.

UpdateAlignmentBox method

See also Example

Updates the alignment box for a shape.

Version added

2000

Syntax

objRet = *object*.**UpdateAlignmentBox**

objRet	Required. An expression that returns a **Shape** object with a new width and height.
object	Required. An expression that returns a **Shape** object.

Remarks

The **UpdateAlignmentBox** method alters the width and height of a shape, often a group. For example, after moving a shape in a group the shape may be outside the group's alignment box. The **UpdateAlignmentBox** method updates the alignment box so it encloses all the shapes in the group.

Note Many shapes are designed so that their alignment boxes don't coincide with their geometric extents. Using the **UpdateAlignmentBox** method on such shapes defeats the intentions of the shape designer.

UpdateUI method

Example

Causes Microsoft Visio to display changes to the user interface represented by a **UIObject** object.

Version added

4.0

Syntax

object.**UpdateUI**

object	Required. An expression that returns a **UIObject** object that represents the user interface that was changed.

Remarks

The **UpdateUI** method updates the Visio user interface with changes made to a **UIObject** object during a session. Use the **CustomMenus** or **CustomToolbars** property of an **Application** object or **Document** object to obtain the **UIObject** object.

XYFromPage method

Example

Transforms a point expressed in the local coordinate system of its **Page** or **Master** object to an equivalent point expressed in the local coordinate system of the **Shape** object.

Version added

2000

Syntax

object.**XYFromPage** *x*, *y*, *xprime*, *yprime*

object	Required. An expression that returns a **Shape** object whose local coordinate system you are transforming the point to.
x	Required **Double**; *x*-coordinate corresponding to *x* in the **Page** or **Master** object's coordinate system.
y	Required **Double**; *y*-coordinate corresponding to *y* in the **Page** or **Master** object's coordinate system.
xprime	Required **Double**; *x*-coordinate in coordinate system of *object.*
yprime	Required **Double**; *y*-coordinate in coordinate system of *object.*

Remarks

The points *x*, *y*, *xprime*, and *yprime* are all treated as internal drawing units.

An exception is raised if *object* is not a **Shape** object of a **Page** or **Master** object.

XYToPage method

Example

Transforms a point expressed in the local coordinate system of a **Shape** object to an equivalent point expressed in the local coordinate system of its **Page** or **Master** object.

Version added

2000

Syntax

object.**XYToPage** *x*, *y*, *xprime*, *yprime*

object	Required. An expression that returns a **Shape** object whose local coordinate system you are transforming the point from.
x	Required **Double**; *x*-coordinate in coordinate system of *shpObj*.
y	Required **Double**; *y*-coordinate in coordinate system of *shpObj*.
xprime	Required **Double**; *x*-coordinate corresponding to *x* in the **Page** or **Master** object's coordinate system.
yprime	Required **Double**; *y*-coordinate corresponding to *y* in the **Page** or **Master** object's coordinate system.

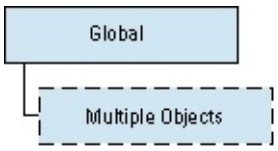
Remarks

The points *x*, *y*, *xprime*, and *yprime* are all treated as internal drawing units.

An exception is raised if *object* is not a **Shape** object of a **Page** or **Master** object.

<Global> object

See also Methods Events



">

The Visio global object is automatically available to Microsoft Visual Basic for Applications (VBA) code that is part of the VBA project of a Visio document. The Visio global object is not available to code in other contexts.

Version added

4.5

Remarks

Members of the global object can be accessed without qualification. For example, to access the **ActivePage** member of the global object:

```
Set pageObj = ActivePage
```

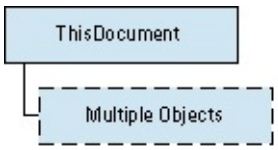
The preceding syntax is different from the syntax you would use for accessing members of non-global objects. For example:

```
Set pageObj = AppObj.ActivePage
```

Note The VBA project of every Visio document also has a class module called **ThisDocument**. When referenced from code in the VBA project, the **ThisDocument** module returns a reference to the project's **Document** object.

ThisDocument object

See also



The Microsoft Visual Basic for Applications (VBA) project of every Microsoft Visio document has a class module called **ThisDocument**. When referenced from code in the project, the **ThisDocument** object returns a reference to the project's **Document** object.

Version added

4.5

Remarks

You can display the name of the VBA project's document in a message box with this statement, for example:

MsgBox ThisDocument.Name

You can get the first page of the VBA project's document by using this code, for example:

```
Dim pagObj As Visio.Page
Set pagObj = ThisDocument.Pages.Item(1)
```

If you want to manipulate the document associated with your VBA project, use the **ThisDocument** object. If you want to manipulate a document, but not necessarily the document associated with your VBA project, get a **Document** object from the **Documents** collection.

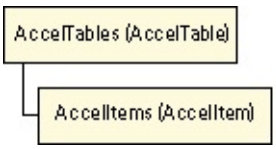
The **ActiveDocument** property often, but not necessarily, returns a reference to the same document as the **ThisDocument** object. The **ActiveDocument** and **ThisDocument** objects are the same if the document shown in the Visio active window is the document containing the **ThisDocument** object's project. Whether your code uses the **ActiveDocument** or **ThisDocument** object depends on the purpose of your program.

You can extend the set of properties and methods of a project's **Document** object by adding public properties and methods to that project's **ThisDocument** class module. The new methods and properties are exposed just like the built-in methods and properties implemented by Visio. The new methods and properties aren't available when you reference other **Document** objects.

Note The **ThisDocument** object is not available to code that isn't part of the VBA project of a Visio document.

AccelItem object

Events



Represents a single accelerator used by Microsoft Visio.

Version added

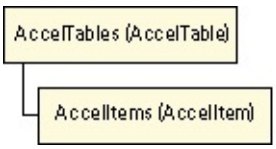
4.0

Remarks

An **AccelItem** object consists of a key, modifiers to the key, and the Visio command or add-on that the accelerator executes when the user presses the accelerator. A key is any ASCII key code, and is not case-sensitive. The modifiers are ALT, CTRL, and SHIFT. Command identifiers are declared by the Visio type library and prefixed with **visCmd**.

AccelItems collection

Events



Includes an **AccelItem** object for each accelerator in a Microsoft Visio window context.

To retrieve an **AccelItems** collection, use the **AccelItems** property of an **AccelTable** object.

Version added

4.0

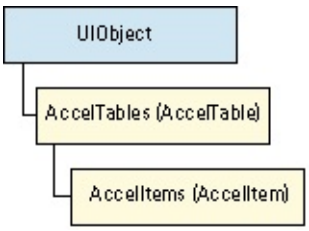
Remarks

The default property for an **AccelItems** collection is **Item**.

Unlike other Visio collections, the **AccelItems** collection is indexed starting with zero (0) rather than 1.

AccelTable object

Events



Represents a Microsoft Windows accelerator table.

Version added

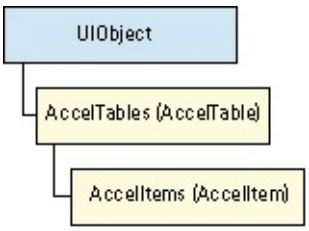
4.0

Remarks

You can create one **AccelTable** object for each Visio window context (drawing window, stencil window, ShapeSheet window, and so forth).

AccelTables collection

Events



Includes an **AccelTable** object for each Microsoft Visio window context that has accelerators.

To retrieve an **AccelTables** collection, use the **AccelTables** property of a **UIObject** object.

Version added

4.0

Remarks

The default property of **AccelTables** is **Item**.

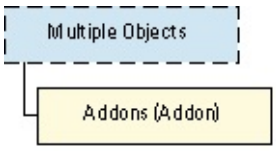
Unlike other Visio collections, the **AccelTables** collection is indexed starting

with zero (0) rather than 1.

An **AccelTable** object is identified in the **AccelTables** collection by its **SetID** property, which corresponds to a Visio window context. For a list of **SetID** values that identify **AccelTable** objects, see the **SetID** property.

Addon object

Events



object;DAR_Objects_(A-M)_1015.htm">

Represents an installed Microsoft Visio add-on.

To retrieve an **Addon** object, use the **Addons** collection of an **Application** object.

Version added

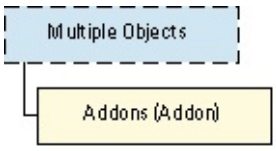
4.0

Remarks

The default property of an **Addon** object is **Name**.

Addons collection

Events



object;DAR_Objects_(A-M)_1015.htm">

Represents the set of installed add-ons known to an **Application** object.

To retrieve an **Addons** collection, use the **Addons** property of an **Application** object.

Version added

4.0

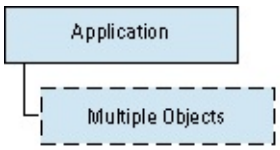
Remarks

The default property of an **Addons** collection is **Item**.

Installed add-ons are those Visio finds in its Addons or StartUp paths, or those that other add-ons have dynamically installed using the **Add** method of the **Addons** collection.

Application (InvisibleApp) object

See also



Represents an instance of Microsoft Visio. An external program typically creates or retrieves an **Application** object before it can retrieve other Visio objects from that instance. Use the Microsoft Visual Basic **CreateObject** function or the **New** keyword to run a new instance, or use the **GetObject** function to retrieve an instance that is already running. You can also use the **CreateObject** function with the **InvisibleApp** object to run a new instance that is invisible. Set the value of the **InvisibleApp** object's **Visible** property to **True** to show it.

Version added

4.1

Remarks

Use the **Documents**, **Windows**, and **Addons** properties of an **Application** object to retrieve the **Document**, **Window**, and **Addon** collections of the instance.

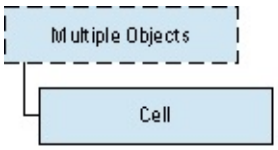
Use the **ActiveDocument**, **ActivePage**, or **ActiveWindow** property to retrieve the currently active **Document**, **Page**, or **Window** object. The **Application** object's menus and toolbars can be accessed using the **BuiltInMenus**, **BuiltInToolbars**, **CustomMenus**, **CustomToolbars**, or **CommandBars** properties.

ActiveDocument is the default property of an **Application** object.

Note Code in the Microsoft Visual Basic for Applications project of a Visio document can use the Visio global object instead of a Visio **Application** object to retrieve other objects.

Cell object

See also



Holds a formula that evaluates to some value.

Version added

2.0

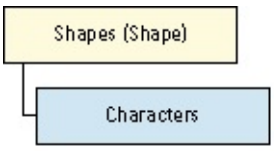
Remarks

The default property of a **Cell** object is **ResultIU**.

You can get or set a cell's formula or value. A cell belongs to a **Shape**, **Style**, or **Row** object and represents a property of the shape, style, or row. For example, the height of a shape equals the value of the shape's Height cell.

A program can control a shape's appearance and behavior by working with the formulas in the shape's cells. You can visually inspect most of a shape's cells by opening the shape's ShapeSheet window. Use the **Cells** or **CellsSRC** property of a **Shape** object to retrieve a **Cell** object. To retrieve a cell in a style, use the **Cells** property of a **Style** object.

Characters object



Represents a shape's text with the text fields expanded to the number of characters they display in a drawing window.

To retrieve a **Characters** object, use the **Characters** property of a **Shape** object.

Version added

3.0

Remarks

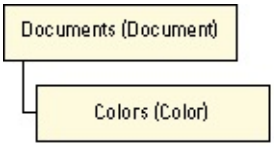
The default property of a **Characters** object is **Text**.

The **Begin** and **End** properties of a **Characters** object determine the range of the shape's text that is represented by the **Characters** object. Initially, the range contains all of the shape's text; you can set the **Begin** and **End** properties to specify a subrange of the text.

After you retrieve a **Characters** object, you can use its **Text** property to retrieve or set the shape's text. Use the **Copy**, **Cut**, or **Paste** method to copy, cut, or paste the **Character** object's text to or from the Clipboard. Use the **CharProps** or **ParaProps** property to change the **Character** object's formatting.

Color object

Methods Events



Represents a color in the color palette for a Microsoft Visio document.

Version added

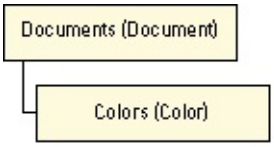
4.0

Remarks

The default property of a **Color** object is **PaletteEntry**.

Colors collection

Methods Events



Includes a **Color** object for each color in the palette for a Microsoft Visio document.

To retrieve a **Colors** collection, use the **Colors** property of a **Document** object.

Version added

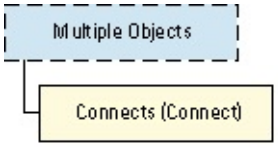
4.0

Remarks

The default property of **Colors** is **Item**.

Connect object

Methods Events



Represents a connection between two shapes in a drawing, such as a line and a box in an organization chart.

Retrieve a **Connect** object from the **Connects** collection returned by the **Connects** and **FromConnects** properties of a **Shape** object, or the **Connects** collection of a **Page** or **Master** object.

Version added

2.0

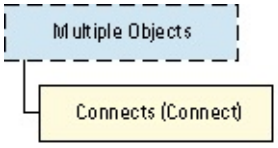
Remarks

The default property of a **Connect** object is **FromSheet**.

Use the **GlueTo** or **GlueToPos** method of a **Cell** object to connect one shape to another in a drawing.

Connects collection

Methods Events



Includes a **Connect** object for each connection between two shapes in a drawing, such as a line and a box in an organization chart.

Version added

2.0

Remarks

The default property of a **Connects** collection is **Item**.

Use the **Connects** property of a **Shape** object to retrieve a **Connects** collection with a **Connect** object for every **Shape** object to which the indicated **Shape** object is connected (glued).

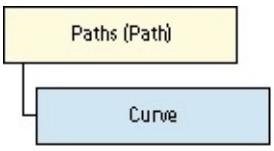
Use the **FromConnects** property of a **Shape** object to retrieve a **Connects** collection with a **Connect** object for every **Shape** object that is connected (glued) to the indicated **Shape** object.

Use the **Connects** property of a **Page** object to retrieve a **Connects** collection with an entry for every connection on the **Page** object.

Use the **Connects** property of a **Master** object to retrieve a **Connects** collection with an entry for every connection in the **Master** object.

Curve object

Events



An item in a **Path** object that represents a consecutive sequence of rows in the Geometry section of its **Path** object.

Version added

5.0

Remarks

The default property of **Curve** object is **Point**.

If a **Curve** object is in a collection returned by the **Paths** property of a **Shape** object, its coordinates are expressed in the shape's parent coordinate system. If the **Curve** object is in a collection returned by the **PathsLocal** property of a **Shape** object, its coordinates are expressed in the shape's local coordinate system. In both cases, the coordinates are expressed in internal drawing units

(inches).

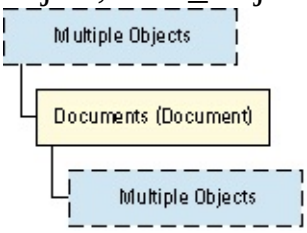
A **Curve** object describes itself in terms of its parameter domain, which is the range [Start(),End()]. Use the **Start** property of a **Curve** object to obtain the curve's starting point and the **End** property of a **Curve** object to obtain the curve's ending point.

Use the **Point** method of a curve object to extrapolate a point along the curve's path. Use the **PointAndDerivatives** method of a **Curve** object to determine a point along the curve's path and, optionally, its first and second derivatives.

Use the **Points** property of a **Curve** object to obtain a stream of points that approximate the curve's path.

Document object

object;DAR_Objects_(A-M)_1015.htm">



Represents a drawing file (.vsd or.vdx), stencil file (.vss or .vsx), or template file (.vst or .vtx) that is open in an instance of Visio. A **Document** object is a member of the **Documents** collection of an **Application** object.

Version added

2.0

Remarks

The default property of a **Document** object is **Name**.

Use the **Open** method of a **Documents** collection to open an existing document.

Use the **Add** method of a **Documents** collection to create a new document.

Use the **ActiveDocument** property of an **Application** object to retrieve the active document in an instance.

Use the **Pages**, **Masters**, and **Styles** properties of a **Document** object to retrieve **Page**, **Master**, and **Style** objects, respectively.

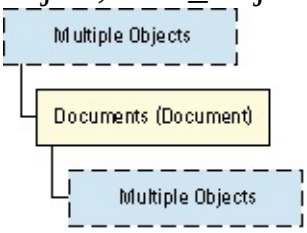
Use the **CustomMenus** or **CustomToolbars** properties of a **Document** object to access the custom menus or toolbars.

Note The Microsoft Visual Basic for Applications (VBA) project of every Visio document also has a class module called **ThisDocument**. When you reference the **ThisDocument** module from code in a VBA project, it returns a reference to the project's **Document** object. For example, the code in a document's project can display the name of the project's document in a message box with this statement:

```
MsgBox ThisDocument.Name
```

Documents collection

object;DAR_Objects_(A-M)_1015.htm">



Includes a **Document** object for each open document in a Microsoft Visio instance.

To retrieve a **Documents** collection, use the **Documents** property of an **Application** object.

Version added

2.0

Remarks

The default property of a **Documents** collection is **Item**.

Event object

Events

A member of the **EventList** collection of a source object such as a **Document**. An event encapsulates an event code.

An **Event** object can trigger two kinds of actions: it can run an add-on, or it can send a notification of the event to the calling program. To create an **Event** object, use the **Add** or **AddAdvise** method of an **EventList** object.

Version added

4.0

Remarks

The default property of an **Event** object is **Event**.

The **Event** property of the **Event** object establishes the event that triggers the action, and its **Action** property indicates the action to be performed.

Use the **Persistable** property to find out if the event can be stored with a Visio document, or the **Persistent** property to find out if the event is stored. Use the **Trigger** method to trigger an **Event** object's action without waiting for the event to occur. Use the **Enabled** property to temporarily disable an event.

EventList collection

Events

Includes an **Event** object for each event to which an object should respond. The object that possesses the **EventList** is sometimes called the source object.

To retrieve an **EventList** collection, use the **EventList** property of the source object.

Version added

4.0

Remarks

The default property of **EventList** is **Item**.

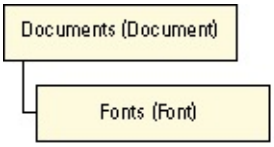
In general, the level of the source object in the Visio object hierarchy determines the scope of its response. For example, if an **Event** object for the **DocumentOpened** event is in the **EventList** of a **Document** object, that event's action is triggered only when that document is opened. If the same **Event** object is in the **EventList** of an **Application** object, the event's action is triggered whenever any document is opened in that instance of Visio.

To create an **Event** object that runs an add-on, use the **Add** method of an **EventList** collection.

To create an **Event** object that sends a notification, use the **AddAdvise** method.

Font object

Methods Events



Represents a typeface that is either applied to text in a document or available for use on the system.

Version added

4.0

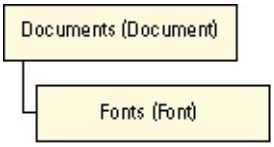
Remarks

The default property of a **Font** object is **Name**.

A **Font** object maps its name (for example, "Arial") to the font ID (for example, 3) that Visio stores in a Font cell in a Character section of a shape whose text is formatted with that font. Font IDs can change when a document is opened on different systems or when fonts are installed or removed.

Fonts collection

Methods Events



Includes a **Font** object for each font applied to text in a document or available on the system.

To retrieve a **Fonts** collection, use the **Fonts** property of a **Document** object.

Version added

4.0

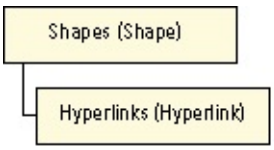
Remarks

The default property of a **Fonts** collection is **Item**.

To retrieve a **Font** object by its font ID (the value shown in the Font cell in a shape's Character section), use the **ItemFromID** property.

Hyperlink object

Events



Completely encapsulates the properties and behavior of a hyperlink. A shape can have one or more hyperlinks that navigate to any named location, such as another page, a local document, or a URL. A **Hyperlink** object enables you to access and manipulate the shape's Hyperlink.Row row.

Version added

5.0

Remarks

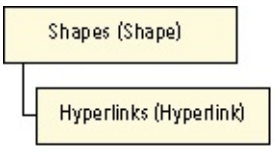
The default property of a **Hyperlink** object is **Description**.

To add a **Hyperlink** object to a shape, use the **AddHyperlink** method.

To navigate to a named hyperlink location, use the **Follow** method.

Hyperlinks collection

Events



Includes **Hyperlink** objects. A **Hyperlinks** collection enables you to access and manipulate a shape's Hyperlinks section.

To retrieve a **Hyperlinks** collection, use the **Hyperlinks** property of a **Shape** object.

Version added

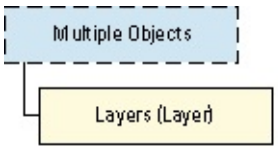
2000

Remarks

The default property of a **Hyperlinks** collection is **Item**.

Layer object

Events



Represents a layer of a page or master. You can assign shapes to or remove them from the layer.

Version added

4.0

Remarks

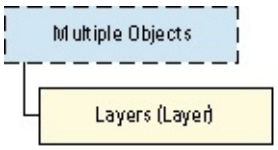
The default property of a **Layer** object is **Name**.

To access cells whose values define layer attributes, such as whether the layer is visible or printable, use the **CellsC** property.

A layer's **Index** and **Row** properties typically have different values. The **Index** property indicates the layer's ordinal position in its **Layers** collection. The layer's **Row** property indicates the index of the row in the Layers section where the layer's attributes are defined.

Layers collection

Events



Includes a **Layer** object for each layer defined for a page or master.

To retrieve a **Layers** collection, use the **Layers** property of a **Page** object or a **Master** object.

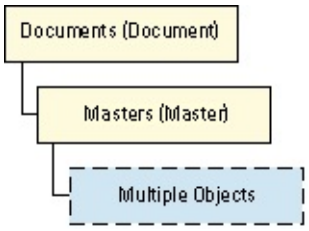
Version added

4.0

Remarks

The default property of **Layers** is **Item**.

Master object



Represents a master in a stencil.

You retrieve a particular **Master** object from the **Masters** collection of a **Document** object whose stencil contains that master.

Version added

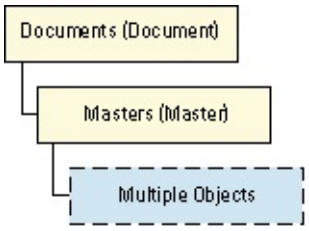
2.0

Remarks

The default property of a **Master** object is **Name**.

To create an instance of a master in a drawing, use the **Drop** method of a **Page** object that represents a drawing page.

Masters collection



Includes a **Master** object for each master in a document's stencil.

To retrieve a **Masters** collection, use the **Masters** property of a **Document** object.

Version added

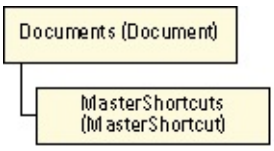
2.0

Remarks

The default property of a **Masters** collection is **Item**.

MasterShortcut object

Events



Represents a master shortcut in a stencil. A master shortcut references a master in a stencil.

Version added

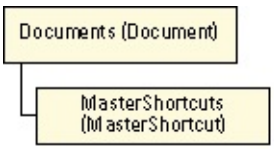
2000

Remarks

The default property of a **MasterShortcut** object is **Name**.

MasterShortcuts collection

Events



Includes **MasterShortcut** objects.

To retrieve a **MasterShortcuts** collection, use the **MasterShortcuts** property of a **Document** object.

Version added

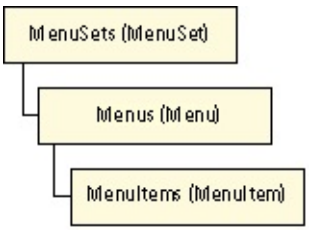
2000

Remarks

The default property of a **MasterShortcuts** collection is **Item**.

Menu object

Events



Represents a single menu on a Microsoft Visio menu bar, such as the **File** menu or **Edit** menu.

Version added

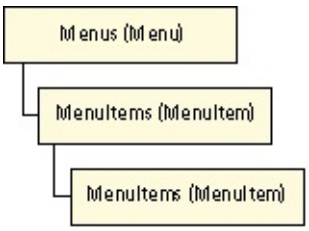
4.0

Remarks

The index of a **Menu** object within the **Menus** collection corresponds to the menu's position from left to right on the menu bar, starting with zero (0) for the menu farthest to the left if the menus are arranged horizontally.

MenuItem object

Events



Represents a single menu item on a Microsoft Visio menu, such as the **Copy** menu item on the **Edit** menu.

Version added

4.0

Remarks

The default property of **MenuItem** is **Caption**.

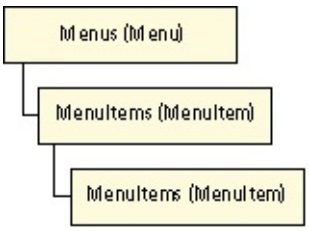
A **MenuItem** object contains all the information it needs to display the menu item and launch the appropriate Visio command or add-on. It also contains text for the **Undo**, **Redo**, and **Repeat** menu items and error messages.

The index of a **MenuItem** object within the **MenuItems** collection corresponds to the menu item's position from top to bottom on the menu or submenu, starting with zero (0).

If the menu item displays a submenu, the **MenuItem** object has a **MenuItems** collection that represents items on the submenu. The **MenuItem** object's **Caption** property contains the submenu title and its **CmdNum** property is set to zero (0). Most of the other properties of the **MenuItem** are ignored, because this object serves much the same role as a **Menu** object.

MenuItems collection

Events



Contains a **MenuItem** object for each command on a Microsoft Visio menu.

To retrieve a **MenuItems** collection, use the **MenuItems** property of a **Menu** object or a **MenuItem** object.

Version added

4.0

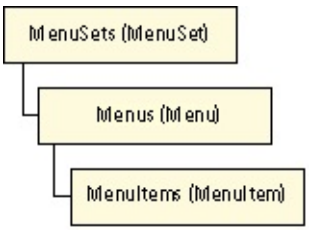
Remarks

The default property of **MenuItems** is **Item**.

Unlike other Visio collections, the **MenuItems** collection is indexed starting with zero (0) rather than 1.

Menus collection

Events



Includes a **Menu** object for each menu in a Microsoft Visio menu set.

To retrieve a **Menus** collection, use the **Menus** property of a **MenuSet** object.

Version added

4.0

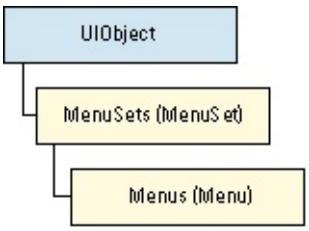
Remarks

The default property of **Menus** is **Item**.

Unlike other Visio collections, the **Menus** collection is indexed starting with zero (0) rather than 1.

MenuSet object

Events



Represents an entire menu set used by a Microsoft Visio window context.

Version added

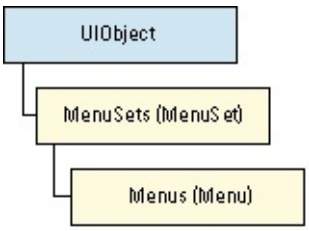
4.0

Remarks

A shortcut menu (which appears when you press the right mouse button) is represented by a **MenuSet** object that has a single untitled **Menu** object in its **Menus** collection, which contains the contents of the shortcut menu in its **MenuItems** collection.

MenuSets collection

Events



Includes a **MenuSet** object for each Microsoft Visio window context that has menus.

To retrieve a **MenuSets** collection, use the **MenuSets** property of a **UIObject** object.

Version added

4.0

Remarks

The default property of **MenuSets** is **Item**.

Unlike other Visio collections, the **MenuSets** collection is indexed starting with

zero (0) rather than 1.

A **MenuSet** object is identified in the **MenuSets** collection by its **SetID** property, which corresponds to a Visio window context. For a list of **SetID** values for **MenuSet** objects, see the **SetID** property.

MSGWrap object

Methods Events

Passed as an argument with the **OnKeystrokeMessageForAddon** event. The **MSGWrap** object wraps the data contained in a message passed from Microsoft Windows to Microsoft Visio.

Version added

2002

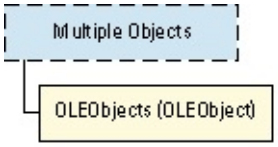
Remarks

The properties of the **MSGWrap** object correspond to the fields in the **MSG** structure defined as part of the Windows operating system.

For details, search for "**MSG** structure" in your Windows documentation or on the Microsoft Developer Network (MSDN) Web site.

OLEObject object

Methods Events



Represents an OLE 2.0 linked or embedded object or an ActiveX control in a Microsoft Visio document, page, or master.

Version added

5.0

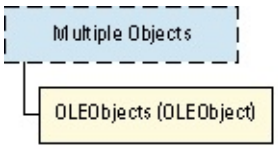
Remarks

The default property of **OLEObject** is **Object**.

To obtain the **IDispatch** interface on an ActiveX control or embedded or linked OLE 2.0 object represented by a shape, use the **Object** property of an **OLEObject** object.

OLEObjects collection

Methods Events



Includes an **OLEObject** object for each OLE 2.0 linked or embedded object or ActiveX control contained in a document, page, or master.

Each member of an **OLEObjects** collection is an **OLEObject** object, which represents an OLE 2.0 linked or embedded object or an ActiveX control in a Microsoft Visio document.

To retrieve an **OLEObjects** collection, use the **OLEObjects** property of a **Document**, **Page**, or **Master** object.

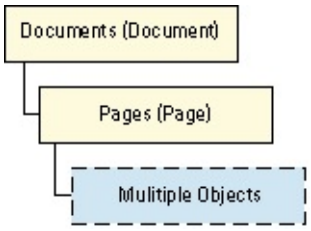
Version added

5.0

Remarks

The default property of **OLEObjects** is **Item**.

Page object



Represents a drawing page, which can be either a foreground page or background page.

Version added

2.0

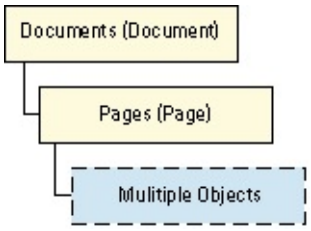
Remarks

The default property of a **Page** object is **Name**.

To retrieve the active page in an instance, use the **ActivePage** property of an **Application** object.

The members of a **Document** object's **Pages** collection represent the pages in that document. To retrieve a page's shapes, use the **Shapes** property of a **Page** object.

Pages collection



Includes a **Page** object for each drawing page in a document.

To retrieve a **Pages** collection, use the **Pages** property of a **Document** object.

Version added

2.0

Remarks

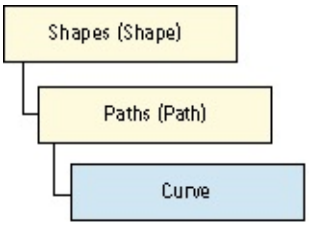
The default property of a **Pages** collection is **Item**.

The order of items in a **Pages** collection is significant: If there are *n* foreground

pages in a document, then the first *n* pages in its **Pages** collection are foreground pages and are in order. The remaining pages in the collection are the background pages of the document; these are in no particular order.

Path object

Methods Events



Represents a sequence of one or more segments whose ends abut. A path describes where a pen would move in order to draw one shape component. Each **Path** object corresponds to a Geometry section of a shape.

Version added

5.0

Remarks

The default property of a **Path** object is **Item**.

A **Curve** object is an item in a **Path** object that is any linear or curved segment representing a consecutive sequence of rows in the Geometry section that the **Path** object represents. The number of **Curve** objects in a **Path** object is not necessarily the same as the number of rows in its Geometry section.

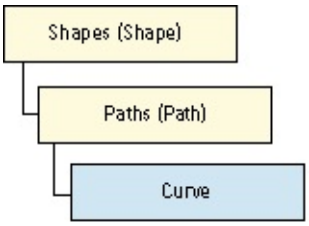
The **Path** object is conceptually of zero width. Line weights, patterns, and ends are ignored, however, corner rounding is included. A **Path** object may or may not be closed, and it may intersect itself. For example, a **Path** may describe a

figure eight.

If you retrieve a **Path** object from a collection obtained by the **Paths** property of a shape, its coordinates are expressed in the shape's parent coordinate system. If you retrieve a **Path** object from a collection obtained by the **PathsLocal** property of a shape, its coordinates are expressed in the shape's local coordinate system. In both cases, coordinates are expressed in internal drawing units (inches).

Paths collection

Methods Events



Includes a **Path** object for each Geometry section for a group or shape.

To retrieve a **Paths** collection expressed in the shape's parent coordinate system, use the **Paths** property of the shape. The coordinates are expressed in internal drawing units (inches).

Version added

5.0

Remarks

The default property of a **Paths** collection is **Item**.

To retrieve a **Paths** collection expressed in the shape's local coordinate system, use the **PathsLocal** property of the shape. The coordinates are expressed in internal drawing units (inches).

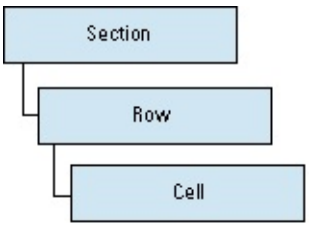
If a **Shape** object is a page, foreign object, or guide, then its **Paths** and **PathsLocal** properties don't contain any items.

If a **Shape** object is a group, then its **Paths** and **PathsLocal** properties are the union of the paths of its component shapes.

If a **Shape** object is a shape, then its **Paths** and **PathsLocal** properties include one item for each Geometry section that defines a stroke of positive length.

Row object

See also



Enables you to access and manipulate a row in a section.

Version added

2000

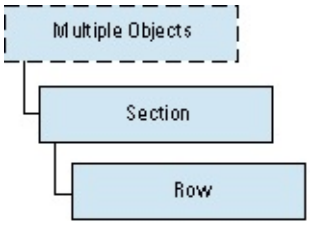
Remarks

The default property of a **Row** object is **Cell**.

Section object

See also

Methods



Enables you to access and manipulate a section of a shape. A section of a shape corresponds to a section shown in the ShapeSheet window. A section contains rows and cells.

Version added

2000

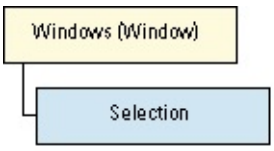
Remarks

The default property of a **Section** object is **Row**.

Selection object

See also

Events



Represents a subset of **Shape** objects for a page or master to which an operation can be applied.

To retrieve a **Selection** object that corresponds to the set of shapes selected in a window, use the **Selection** property of a **Window** object.

Version added

2.0

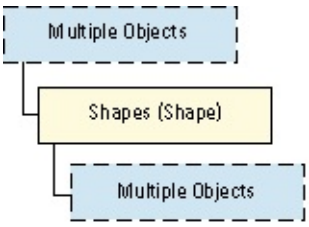
Remarks

The default property of a **Selection** object is **Item**.

After you retrieve a **Selection** object, you can add or remove shapes by using the **Select** method.

By default, the items reported by a **Selection** object do not include subselected or superselected **Shape** objects. Use the **IterationMode** property to control whether subselected and superselected **Shape** objects are reported. You can determine whether an individual item is subselected or superselected using the **ItemStatus** property.

Shape object



Represents anything you can select in a drawing window: a basic shape, a group, a guide, or an object from another application embedded or linked in Microsoft Visio.

Version added

2.0

Remarks

The default property of a **Shape** object is **Name**.

You can retrieve a particular **Shape** object from the **Shapes** collection of the following objects:

Page object

Master object

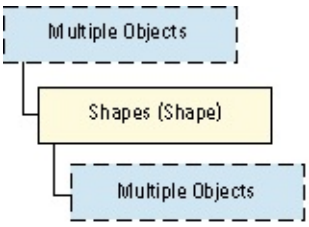
Shape object that represents a group

To retrieve **Cell** objects and **Connect** objects, use the **Cells** and **Connects** properties of a **Shape** object, respectively.

Note The **PageSheet** property of a **Page** object and **Master** object returns a **Shape** object whose **Type** property returns **visTypePage**. It has cells that specify properties such as drawing size and drawing scale. The **DocumentSheet** property of a **Document** object also returns a **Shape** object whose **Type** property returns **visTypeDoc**. It has cells that specify properties of the document.

Shapes collection

Events



Includes a **Shape** object for each basic shape, group, guide, or object from another application (linked or embedded in Microsoft Visio) on a drawing page, master, or group.

Version added

2.0

Remarks

To retrieve a **Shapes** collection, use the **Shapes** property of a **Page**, **Master**, or **Shape** object.

The default property of a **Shapes** collection is **Item**.

The order of items in a **Shapes** collection corresponds to the stacking (drawing) order of the shapes.

StatusBar object

See also Properties Methods Events

Beginning with Microsoft Visio 2002, this object is obsolete.

Remarks

In earlier versions, this represented a status bar shown at the bottom of a Visio window.

StatusBarItem object

See also Properties Methods Events

Beginning with Microsoft Visio 2002, this object is obsolete.

Remarks

In earlier versions, this represented a single item (button, message, and so forth) on a status bar.

StatusBarItems collection

Beginning with Microsoft Visio 2002, this collection is obsolete.

Remarks

In earlier versions, this collection included a **StatusBarItem** object for each Visio window context.

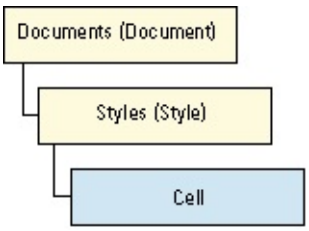
StatusBars collection

Beginning with Microsoft Visio 2002, this collection is obsolete.

Remarks

In earlier versions, this included a **StatusBar** object for each Visio window context that could display a status bar.

Style object



Represents a style defined in a document.

You retrieve a particular style from the **Styles** collection of a **Document** object.

Version added

2.0

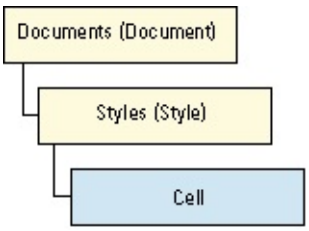
Remarks

The default property of a **Style** object is **Name**.

Any **Shape** object to which a style is applied inherits the attributes defined by the style. Use the **LineStyle**, **FillStyle**, **TextStyle**, or **Style** property of a **Shape** object to apply a style to a shape or to determine what style is applied to a shape.

Like a **Shape** object, a **Style** object has cells whose formulas define the values of the style's attributes. To retrieve one of these cells, use the **Cells** or **CellsSRC** property of the **Style** object.

Styles collection



Includes a **Style** object for each style defined in a document.

Version added

2.0

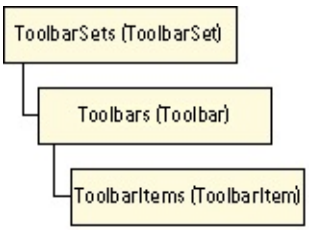
Remarks

To retrieve a **Styles** collection, use the **Styles** property of a **Document** object.

The default property of a **Styles** collection is **Item**.

Toolbar object

Events



Represents a group of toolbar items in a Microsoft Visio window.

Version added

4.0

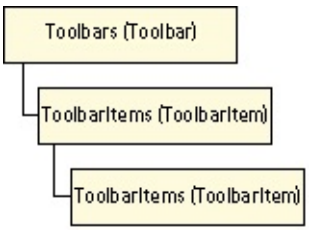
Remarks

The default property of **Toolbar** is **Caption**.

The index of the **Toolbar** object within the **Toolbars** collection corresponds to its order in the Visio window, starting with zero (0) for the toolbar closest to the top. Up to 10 toolbars can be displayed in a Visio window at one time.

ToolbarItem object

Events



Represents one item in a **Toolbar** object. A **ToolbarItem** object can represent a button, combo box, or any other item on the Microsoft Visio toolbars.

Version added

4.0

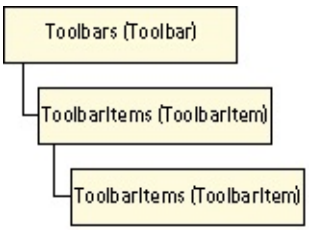
Remarks

The index of the **ToolbarItem** object within the **ToolbarItems** collection corresponds to its position on the toolbar, starting with zero (0) for the item farthest to the left if the toolbars are arranged horizontally.

Beginning with Microsoft Visio 2002, use the **BeginGroup** property to create spaces on a toolbar.

ToolbarItems collection

Events



Includes a **ToolbarItem** object for each item on a toolbar.

Version added

4.0

Remarks

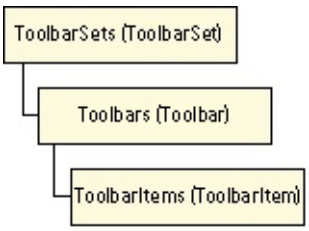
To retrieve a **ToolbarItems** collection, use the **ToolbarItems** property of a **Toolbar** object.

The default property of **ToolbarItems** is **Item**.

Unlike other Visio collections, the **ToolbarItems** collection is indexed starting with zero (0) rather than 1.

Toolbars collection

Events



Includes a **Toolbar** object for each toolbar in a window context.

Version added

4.0

Remarks

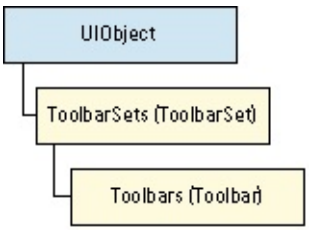
To retrieve a **Toolbars** collection, use the **Toolbars** property of a **ToolbarSet** object.

The default property of **Toolbars** is **Item**.

Unlike other Visio collections, the **Toolbars** collection is indexed starting with zero (0) rather than 1.

ToolbarSet object

Events



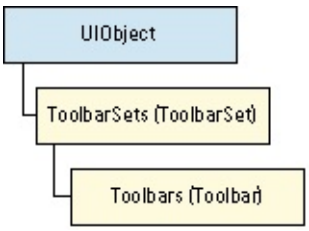
Represents the set of toolbars for a Microsoft Visio window context.

Version added

4.0

ToolbarSets collection

Events



Includes a **ToolbarSet** object for each window context that can display toolbars.

Version added

4.0

Remarks

To retrieve a **ToolbarSets** collection, use the **ToolbarSets** property of a **UIObject** object.

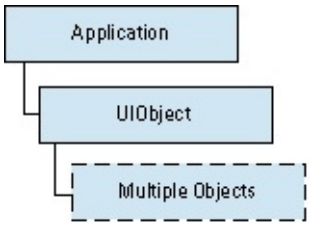
The default property of **ToolbarSets** is **Item**.

Unlike other Visio collections, the **ToolbarSets** collection is indexed starting with zero (0) rather than 1.

A **ToolbarSet** object is identified in the **ToolbarSets** collection by its **SetID** property, which corresponds to a Visio window context. For a list if **SetID** values for **ToolbarSet** objects, see the **SetID** property.

UIObject object

Events



Represents a set of Microsoft Visio menus, toolbars, and accelerators, from either the built-in Visio user interface or a customized version of it.

Version added

4.0

Remarks

To retrieve a **UIObject** object that contains

Visio menus and accelerators, use the **BuiltInMenus** property of an **Application** object and then the **Menusets** or **AccelTables** collections of the **UIObject**

object returned from the **BuiltInMenus** property.

Visio toolbars, use the **BuiltInToolbars** property of an **Application** object and then the **ToolbarSets** collection of the **UIObject** object returned from the **BuiltInToolbars** property.

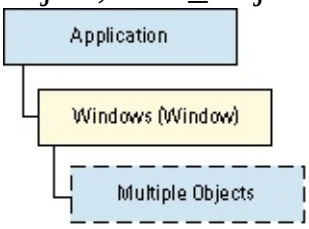
If an **Application** object or **Document** object has a customized user interface, use the **CustomMenus** or **CustomToolbars** properties to retrieve **UIObject** objects that represent these.

A **UIObject** object can be stored in a file and loaded into Visio. Use the **SaveToFile** method to save the object and the **LoadFromFile** method to load it, or set the **CustomMenusFile** or **CustomToolbarsFile** property of an **Application** object or **Document** object to the name of the stored user interface file.

Beginning with Microsoft Visio 2002, a program can manipulate menus and toolbars in the Visio user interface by manipulating the **CommandBars** collection returned by the **CommandBars** property. The **CommandBars** collection has an interface identical to the **CommandBars** collection exposed by the suite of Microsoft Office applications such as Microsoft Word and Microsoft Excel. Consequently, programs can manipulate the Visio menus and toolbars using either the **CommandBars** collection or **UIObject** objects.

Window object

object;DAR_Objects_(A-M)_1015.htm">



Represents an open window in a Microsoft Visio instance.

Version added

2.0

Remarks

The default property of a **Window** object is **Application**.

To retrieve

the active window in an instance of Visio, use the **ActiveWindow** property of an **Application** object.

a **Page** object that represents the page shown in the window, use the **Page** property of a **Window** object.

a **Document** object that represents the document displayed in that window, use the **Document** property.

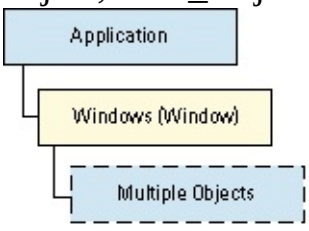
a **Selection** object that represents the shapes selected in that window, use the **Selection** property.

Note Beginning with Microsoft Visio 2002, the following methods of the **Window** object are obsolete: **AddToGroup**, **Cut**, **Combine**, **Copy**, **Delete**, **Duplicate**, **Fragment**, **Group**, **Intersect**, **Join RemoveFromGroup**, **Subtract**, **Trim**, and **Union**. Existing solutions that invoke these methods will continue to work properly; however, new or rebuilt solutions should use these methods with the **Selection** object.

In addition, the **Window** object's **Paste** method is now obsolete. Use the **Paste** or **PasteSpecial** method of the **Page**, **Master**, or **Shape** object. (Use the **Shape** object in the case of group shapes.)

Windows collection

object;DAR_Objects_(A-M)_1015.htm">



Includes a **Window** object for a window that is open in the application.

Version added

2.0

Remarks

To retrieve a **Windows** collection, use the **Windows** property of an **Application** object.

The default property of a **Windows** collection is **Item**.

If a docked stencil window contains more than one stencil, only one window is counted.

AccelItems property

See also

Returns the **AccelItems** collection of an **AccelTable** object.

Version added

4.0

Syntax

objRet = object.**AccelItems**

objRet	An **AccelItems** collection.
object	Required. An expression that returns an **AccelTable** object.

AccelTables property

Returns the **AccelTables** collection of a **UIObject** object.

Version added

4.0

Syntax

objRet = object.**AccelTables**

objRet	An **AccelTables** collection.
object	Required. An expression that returns the **UIObject** object that owns the collection.

Remarks

If a **UIObject** object represents menu items and accelerators (for example, if you retrieved the object using the **BuiltInMenus** property of an **Application** object), then its **AccelTables** collection represents tables of accelerator keys for that **UIObject** object.

To retrieve accelerators for a particular window context, for example, the drawing window, use the **ItemAtID** property of an **AccelTables** collection. If a window context does not include accelerators, it has no **AccelTables** collection. For a list of valid window context IDs, see the **SetID property**.

Action property

Example

Gets or sets the action code of an **Event** object.

Version added

4.0

Syntax

intRet = *object*.**Action**

object.**Action** = *actionCode*

intRet	**Integer**. The **Event** object's action code.
object	Required. An expression that returns an **Event** object.
actionCode	Required **Integer**. The new action code to assign.

Remarks

An **Event** object consists of an event-action pair—an event triggers an action. An action code is the numeric constant for the action that the event triggers.

Visio supports the following action codes.

Constant	Value
visActCodeRunAddon	1
visActCodeAdvise	2

ActionText property

Gets or sets the action text for a menu, menu item, or toolbar item.

Version added

4.0

Syntax

object.**ActionText** = *actionStr*

actionStr = *object*.**ActionText**

object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** object that owns the action text.
actionStr	Required **String**. A string that describes the action.

Remarks

Action text is a string that describes the action on the **Undo**, **Redo**, and **Repeat** menu items on the **Edit** menu.

If the **ActionText** property is empty and the object's **CmdNum** property is set to one of the Visio built-in command IDs, the item uses the default action text from the built-in Visio user interface.

Active property

See also

Indicates whether the instance of Microsoft Visio represented by the **Application** object is the active application on the Microsoft Windows desktop —the application with the highlighted title bar.

Version added

4.1

Syntax

intRet = object.**Active**

intRet	**Integer**. **False** (0) if the application is not active; **True** (-1) if it is active.
object	Required. An expression that returns an **Application** object.

Remarks

The active application on the Windows desktop is distinct from the active Visio instance, which is returned by a call to the OLE **GetActiveObject** function (**GetObject** function in Microsoft Visual Basic). The **GetObject** function

retrieves the instance of Visio that was most recently activated, which may or may not be the active application on the desktop at that moment. Of all instances of Visio that are currently running, only one is the active Visio instance.

For example, suppose you start one instance of Visio and one of another application, such as Microsoft Excel.

If the instance of Visio is the active application on your desktop, **GetObject**(, "visio.application") retrieves that instance and its **Active** property is **True**.

If you activate the instance of Microsoft Excel, **GetObject**(, "visio.application") retrieves the same instance of Visio, but its **Active** property is **False**.

If an **Application** object's **Active** property is **True**, you can assume that the corresponding instance of Visio is the active instance of Visio unless the **InPlace** property is also **True**. If an instance of Visio is activated for in-place editing in a container application, that instance may not necessarily report itself as the active instance of Visio.

ActiveDocument property

object;DAR_Objects_(A-M)_1015.htm">

Returns the active **Document** object, the document shown in the active window.

Version added

2.0

Syntax

objRet = *object*.**ActiveDocument**

objRet	A **Document** object that represents the active document.
object	Required. An expression that returns the **Application** object that owns the document.

Remarks

When no documents are open, there is no active document and the **ActiveDocument** property returns the value **Nothing** and does not raise an exception.

If your code is in the Microsoft Visual Basic for Applications (VBA) project of a Visio document, the **ActiveDocument** property often, but not necessarily, returns a reference to the **ThisDocument** object, a class module in the VBA project of every Visio document. If the **ThisDocument** object is shown in the active window, then the **ActiveDocument** object and the **ThisDocument** object refer to the same document. When the **ThisDocument** object is referenced from

code in a project, it returns a reference to the project's **Document** object.

Whether you use the **ActiveDocument** object or the **ThisDocument** object depends on the purpose of your code.

You can compare the result returned by the **ActiveDocument** property with the value **Nothing** to determine if a document is active. If the value of the **Application.Documents.Count** property is greater than zero, then at least one document is open and active.

ActivePage property

object;DAR_Objects_(A-M)_1015.htm">

Returns the active **Page** object.

Version added

2.0

Syntax

objRet = object.**ActivePage**

objRet	A **Page** object that represents the active page.
object	Required. An expression that returns the **Application** object that owns the page.

Remarks

The **ActivePage** property returns a **Page** object only when the active window displays a drawing page; otherwise, it returns **Nothing**. To verify that a page is active, use the **Is** operator to compare the **ActivePage** property with **Nothing**.

ActivePrinter property

Example

Specifies the printer that all Microsoft Visio documents print to.

Version added

2002

Syntax

strRet = *object*.**ActivePrinter**

object.**ActivePrinter** = *strExpression*

strRet	**String**. The current active printer.
object	Required. An expression that returns an **Application** object.
strExpression	Required **String**. The new active printer.

Remarks

The **ActivePrinter** property is initially set to the default printer.

ActiveWindow property

object;DAR_Objects_(A-M)_1015.htm">

Returns the active **Window** object.

Version added

2.0

Syntax

objRet = object.**ActiveWindow**

objRet	A **Window** object that represents the active window.
object	Required. An expression that returns the **Application** object that owns the window.

Remarks

The active window can be one of the following window types: Drawing, Stencil, ShapeSheet, Edit Icon, or a Drawing or Stencil window created by an add-on. The application's active window can only be an MDI frame window—it cannot be one of the floating, docked, or anchored windows. For a complete list of window types, see the **Type** property.

If a window in an instance of Visio is not active, the **ActiveWindow** property returns **Nothing**.

Addins property

See also Example Applies to

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions of Visio, this property returned a collection of registered COM add-ins. Use the **COMAddins** property to get a reference to the **COMAddins** collection.

AddonArgs property

Gets or sets the argument string that you send to the add-on associated with a particular menu or toolbar.

Version added

4.0

Syntax

object.**AddonArgs** = *argsStr*

argsStr = *object*.**AddonArgs**

object	Required. An expression that returns an object in the **Applies to** list that starts the add-on.
argsStr	Required **String**. The argument string to be passed to the add-on.

Remarks

An argument's string can be anything appropriate for the add-on. However, the

arguments are packaged together with other information into a command string, which cannot exceed 127 characters. For best results, limit arguments to 50 characters.

An object's **AddonName** property indicates the name of the add-on to which the arguments are sent.

AddonName property

Gets or sets the name of an add-on associated with a menu or toolbar.

Version added

4.0

Syntax

object.**AddonName** = *addonStr*

addonStr = *object*.**AddonName**

object	Required. An expression that returns an object in the **Applies to** list.
addonStr	Required **String**. The name of the add-on to be run or Microsoft Visual Basic for Applications (VBA) code to be executed.

Remarks

When an item whose **AddonName** property is set is selected, Visio asks the

VBA project of the active document to parse the **AddonName** property string. If VBA successfully parses the string, Visio tells VBA to execute the string. Using this technique, you can cause a menu or toolbar item to run a VBA macro or procedure, show a VBA form, log information to the Immediate window, and so on. See the **ExecuteLine** method for examples.

If VBA cannot parse the string, then Visio runs the add-on named by the **AddonName** property. If there is no such add-on, Visio does nothing.

If the **AddonName** property is set, Visio ignores the object's **CmdNum** property.

Use the **AddonArgs** property to specify arguments to send to the add-on when it is run.

AddonPaths property

Example

Gets or sets the paths where Microsoft Visio looks for add-ons.

Version added

4.0

Syntax

strRet = *object*.**AddonPaths**

object.**AddonPaths** = *strPaths*

strRet	**String**. A list of folders.
object	Required. An expression that returns an **Application** object.
strPaths	Required **String**. A text string containing a list of folders. Use semicolons to separate individual folders in the string.

Remarks

To indicate more than one folder in the path where Visio looks for add-ons, separate individual items in the path string with semicolons.

The string passed to and received from the **AddonPaths** property is the same string shown on the **File Paths** tab in the **Options** dialog box (click **Options** on the **Tools** menu). This string is stored in **HKEY_CURRENT_USER\Software\Microsoft\Visio\application\AddonsPat**

When Visio looks for add-ons, it looks in all paths named in the **AddonPaths** property and all the subfolders of those paths. If you pass the **AddonPaths** property to the **EnumDirectories** method, it returns a complete list of fully qualified paths in which Visio looks.

If a path is not fully qualified, Visio looks for the folder in the folder that contains the Visio program files (*appObj*.**Path**). For example, if the Visio executable file is installed in c:\Visio, and the **AddonPaths** property is "Add-ons;d:\Add-ons", Visio looks for add-ons in both c:\Visio\Add-ons and d:\Add-ons.

Addons property

object;DAR_Objects_(A-M)_1015.htm">

Returns the **Addons** collection of an **Application** object.

Version added

4.0

Syntax

objRet = object.**Addons**

objRet	The **Addons** collection of the **Application** object.
object	Required. An expression that returns the **Application** object that owns the collection.

Remarks

The **Addons** collection includes an **Addon** object for each add-on in the folders specified by the **AddonPaths** property and for each add-on that is added dynamically to the collection by other add-ons.

Address property

Gets or sets the address for a shape's **Hyperlink** object—the address to which the hyperlink navigates.

Version added

5.0

Syntax

strRet = *object*.**Address**

object.**Address** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Hyperlink** object.
stringExpression	Required **String**. The new value for the field.

Remarks

Setting the **Address** property for a **Hyperlink** object is equivalent to entering information in the **Address** box in the **Hyperlinks** dialog box (click **Hyperlinks**

on the **Insert** menu), or setting the result of the Address cell in the shape's Hyperlink.Row row through the ShapeSheet window.

The **Address** property value can be a DOS, UNC, or URL path, for example, c:\Drawings\MyDrawing.vsd, \\Server\Shared\MyDrawing.vsd, or http://www.microsoft.com, respectively.

If the **Address** property is relative, for example, "..\Drawing.vsd", then it is composed against the **HyperlinkBase** property, if supplied, or the hyperlink's document path. If the document is not saved, the hyperlink is undefined.

If the **Address** property is empty, then you can assume the address points to a page in the document that contains the page. In this case, the **SubAddress** property contains the name of the drawing page to which the hyperlink navigates.

AlertResponse property

Example

Determines whether Microsoft Visio shows alerts and modal dialog boxes to the user.

Version added

4.1

Syntax

intRet = *object*.**AlertResponse**

object.**AlertResponse** = *intExpression*

intRet	**Integer**. Zero (0) to display alerts to the user and allow the user to respond, or the value of the default response (see Remarks).
object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. Zero (0) to display alerts to the user and allow the user to respond, or the value of the default response to supply (see Remarks).

Remarks

Certain operations, such as closing a document with unsaved modifications, cause Visio to display an alert or modal dialog box requesting the user to supply a response such as **OK**, **Yes**, **No**, or **Cancel**. To prevent Visio from displaying alerts or modal dialog boxes when a program performs such actions, set the **AlertResponse** property to a default value for the response. In this case, Visio does not display the alert or modal dialog box; instead, Visio behaves as if the user responded to the alert or modal dialog box with the value of the **AlertResponse** property.

If the **AlertResponse** property is 0 (its default value), alerts and modal dialog boxes are displayed.

The values you supply for the **AlertResponse** property correspond to the standard Windows constants IDOK, IDCANCEL, and so forth.

Constant	Value
IDOK	1
IDCANCEL	2
IDABORT	3
IDRETRY	4
IDIGNORE	5
IDYES	6
IDNO	7

AlignName property

See also Example

Gets or sets the position of a master name in a stencil window.

Version added

2.0

Syntax

intRet = *object*.**AlignName**

object.**AlignName** = *intNewAlignment*

intRet	**Integer**. Returns the current alignment of the master's name.
object	Required. An expression that returns a **Master** object.
intNewAlignment	Required **Integer**. The new alignment for the master's name.

Remarks

The following constants declared by the Visio type library show the possible alignment values.

Constant	Value
visLeft	1
visCenter	2
visRight	3

AllowEditing property

See also Example

Determines whether the **Edit** command is enabled or disabled in a stencil window.

Version added

2002

Syntax

boolRet = *object*.**AllowEditing**

object. **AllowEditing** = *boolValue*

boolRet	**Boolean**. **True** if the **Edit** command is enabled in a stencil window; **False** if it is disabled.
object	Required. An expression that returns a **Window** object.
boolValue	Required **Boolean**. **True** to enable the **Edit** command; **False** to disable it.

Remarks

Use the **AllowEditing** property to prevent unintentional editing in the stencil.

Alt property

Example

Determines whether the ALT key is a modifier for an accelerator.

Version added

4.0

Syntax

intRet = *object*.**Alt**

object.**Alt** = *intExpression*

intRet	**Integer**. **True** (-1) if the ALT key modifies an accelerator key in an **AccelItem** object; otherwise, **False** (0).
object	Required. An expression that returns an **AccelItem** object.
intExpression	Required **Integer**. **True** (non-zero) if the ALT key modifies an accelerator key in an **AccelItem** object; otherwise, **False** (0).

AlternateNames property

See also

Gets or sets the alternate names for a document.

Version added

2000

Syntax

strRet = *object*.**AlternateNames**

object.**AlternateNames** = *strExpression*

strRet	**String**. One or more file names delimited by semicolons.
object	Required. An expression that returns a **Document** object.
strExpression	Required **String**. One or more file names delimited by semicolons. For example, "My Shapes 99.vss"; "My Shapes 98.vss".

Remarks

The application stores document names in the following situations:

Templates store stencil names. For example, the **Basic Flowchart** template stores the names of the **Basic Flowchart Shapes.vss** and **Backgrounds.vss** stencils. These stencils are opened with the **Basic Flowchart** template.

Master shortcuts store stencil names. For example, a shortcut for the **Data** shape stores the name of the stencil on which the **Data** shape is stored—**Basic Flowchart Shapes.vss**.

When the application opens a document or accesses the **Document** object's collection, it uses the document name. If Visio can't find the document name, it looks for alternate names for those stencils that are in the correct path (on the **Tools** menu, click **Options**, and then click the **File Paths** tab to add a path). For example, suppose you created the stencil named "My Shapes 98.vss." The following year you revised the stencil and renamed it "My Shapes 99.vss." Any templates that opened **My Shapes 98.vss** should now open **My Shapes 99.vss**. To do this, set the **AlternateNames** property of **My Shapes 99.vss** to "My Shapes 98.vss." The following VBA code shows one way to do this:

```
Visio.Documents("My Shapes 99.vss").AlternateNames =
```

The **AlternateNames** property is empty until you set it through Automation. Each of the alternate names in the string should contain the file name, with no folder information. You can also include comments in angle brackets (<>) as the application ignores anything in angle brackets. For example, you could set the **AlternateNames** property like this:

```
Visio.Documents("HRShapes.vss").AlternateNames = "Hu
```

Application property

See also object;DAR_Objects_(A-M)_1015.htm">

Returns the instance of Microsoft Visio that is associated with an object.

Version added

2.0

Syntax

objRet = *object*.**Application**

objRet	The **Application** object that is associated with an object.
object	Required. An expression that returns an object in the **Applies to** list.

AreaIU property

Example

Returns the area of an object in internal units (square inches).

Version added

4.0

Syntax

retVal = object.**AreaIU**

retVal	**Double**. The area of the object in internal units.
object	Required. An expression that returns a **Shape** object.

Attributes property

See also Example

Returns the attributes of the font.

Version added

3.0

Syntax

intRet = *object*.**Attributes**

intRet	**Integer**. The attributes of a **Font** object.
object	Required. An expression that returns a **Font** object.

Remarks

When using the **Attributes** property with a **Font** object, one of the following values is returned.

Constant	Value
visFontRaster	16
visFontDevice	32
visFontScalable	64
visFont0Alias	128

A font marked as the font 0 alias is used instead of font 0 (the default font). The

font 0 alias is used in some localized versions of Visio and is controlled through entries in the registry.

AutoLayout property

Example

Allows you to temporarily disable the action of the automatic layout functionality, and then reenable it after you are finished with an action.

Version added

2000

Syntax

boolRet = *object*.**AutoLayout**

object.**AutoLayout** = *boolValue*

boolRet	**Boolean**. **True** if automatic layout is enabled, **False** if it is disabled.
object	Required. An expression that returns an **Application** object.
boolValue	Required **Boolean**. **True** to enable automatic layout; **False** to disable automatic layout.

Remarks

Using the **AutoLayout** property helps to improve the performance of add-ons that execute many operations in connected drawings that use Visio automatic layout functionality.

AutoRecover property

Determines whether an open document with unsaved changes is copied when automatic recovery is enabled.

Version added

2000 SR-1

Syntax

boolRet = *object*.**AutoRecover**

object.**AutoRecover** = *boolValue*

boolRet	**Boolean**. **True** if automatic recovery is enabled, **False** if it is disabled.
object	Required. An expression that returns a **Document** object.
boolValue	Required **Boolean**. **True** to enable automatic recovery; **False** to disable automatic recovery.

Remarks

If automatic recovery is enabled (the **AutoRecoverInterval** property is greater than 0), all documents that are open and have unsaved changes are copied into temporary files. If you do not want a document to be recovered, set its **AutoRecover** property to **False**. The **AutoRecover** property is not saved with a document and must be set each time the document opens.

When Visio launches after an abnormal termination and determines that automatic recovery was enabled, it attempts to open all files that were open at termination.

If there is a recovery file that is more recent than the last saved copy of the file, it opens the recovered file and displays the name "<file name> (Recovered)" in the document title bar.

If there is no recovery file, Visio opens the last saved copy of the document.

You must still save changes to recovered documents before Visio closes. If recovered documents are not saved, changes will be deleted as in any unsaved document.

AutoRecoverInterval property

Represents the time interval (in minutes) for how often you want to create copies of open documents with unsaved changes in case of a power failure or an application error.

Version added

2000 SR-1

Syntax

intRet = *object*.**AutoRecoverInterval**

object.**AutoRecoverInterval** = *intExpression*

intRet	**Integer**. An integer value from zero (0) to 120 representing the interval in minutes. The default is 0.
object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. An integer value from 0 to 120 representing the interval in minutes. The default is 0.

Remarks

If the value of the **AutoRecoverInterval** property is less than or equal to 0, no automatic recovery copies are created.

If the value of the **AutoRecoverInterval** property is greater than 0, automatic recovery is enabled for all documents in the Visio instance. To disable automatic recovery for a particular document, set its **AutoRecover** property to **False**.

AvailablePrinters property

Example

Returns a list of installed printers.

Version added

2002

Syntax

strRet = object.**AvailablePrinters**

strRet	**String**. An array of printers installed on the computer.
object	Required. An expression that returns an **Application** object.

Background property

Determines whether a page is a background page.

Version added

2.0

Syntax

retVal = object.**Background**

object.**Background** = *intExpression*

retVal	Integer. **True** if the page is a background page; otherwise, **False**.
object	Required. An expression that returns a **Page** object.
intExpression	Required **Integer**. **False** (0) to declare the page as a foreground page; **True** (non-zero) to declare it as a background page.

BackPage property

Example

Gets or sets the background page of a page.

Version added

2.0

Syntax

objVariantRet = *object*.**BackPage**

object.**BackPage** = *stringVariant*

objVariantRet	**Variant**. A **Page** object that represents the background page.
object	Required. An expression that returns a **Page** object.
stringVariant	Required **Variant.** A string that names the new background page.

Remarks

If a page has no background, its **BackPage** property returns an empty **Variant**. Otherwise the returned **Variant** refers to a **Page** object—the background page of

the indicated page.

To assign a background page to a page, set the page's **BackPage** property to the name of the background page you want to assign. To cause a page to have no background page, pass an empty string to the **BackPage** property.

Note In earlier versions of Visio (through version 4.1), the **BackPage** property returned an object (as opposed to a **Variant** of type object) and it accepted a string (as opposed to a **Variant** of type string). The property has been modified so that it accepts and returns variants due to changes in Automation support tools. For backward compatibility, the **BackPageAsObj** and **BackPageFromName** properties have been added. These properties have the same signatures and occupy the same vtable slots as the earlier version of the **BackPage** property.

BackPageAsObj property

See also Example Applies to

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property returned the background page of a page.

BackPageFromName property

See also Example Applies to

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property set the background page of a page.

BasedOn property

Example

Gets or sets the style on which a **Style** object is based.

Version added

4.0

Syntax

strVal = *object*.**BasedOn**

object.**BasedOn** = *styleName*

strVal	**String**. The name of the current style.
object	Required. An expression that returns a **Style** object.
styleName	Required **String**. The name of the new style.

Remarks

To base a style on no style, set the **BasedOn** property to a zero-length string ("").

BaseID property

Example

Returns a base ID for a master.

Version added

2000

Syntax

```
strRet = object.BaseID
```

strRet	**String**. A **Master** object's base ID.
object	Required. An expression that returns a **Master** object.

Remarks

A base ID is assigned to a master when it is created. When a master is copied, the copies all have the same base ID as the original master.

A **Master** object also has a **UniqueID** property that remains the same as the original master when copied. If the copy of the master gets changed, the unique ID changes but the base ID remains the same.

The only way to change a master's base ID is to use the **NewBaseID** property.

Begin property

Example

Gets or sets the beginning index of a **Characters** object, which represents a range of text in a shape.

Version added

3.0

Syntax

intRet = *object*.**Begin**

object.**Begin** = *intExpression*

intRet	**Integer**. The current beginning index of the **Characters** object.
object	Required. An expression that returns a **Characters** object.
intExpression	Required **Integer**. The new beginning index of the **Characters** object.

Remarks

The **Begin** property determines the beginning of the text range represented by a

Characters object. The value of the **Begin** property is an index that represents the boundary between two characters, similar to an insertion point in text. Like selected text in a drawing window, a **Characters** object represents the sequence of characters that are affected by subsequent actions, such as the **Cut** or **Copy** method. When you retrieve a **Characters** object, its current text range includes all the shape's text. You can change the text range by setting the **Characters** object's **Begin** and **End** properties. Changing the text range of a **Characters** object has no effect on the text of the corresponding shape.

The **Begin** property can have a value from zero (0) to the value of the **CharCount** property for the corresponding shape. An index of 0 is before the first character in the shape's text. An index that is the same as the **CharCount** property is after the last character in the shape's text. If you specify a value less than 0, Visio uses 0. If you specify a value that is inside the expanded characters of a field, Visio sets the value of the **Begin** property to the start of the field.

The value of the **Begin** property must always be less than or equal to the value of the **End** property. If you attempt to set the value of the **Begin** property to a value greater than the **End** property, Visio sets both the **Begin** and **End** properties to the value specified for the **Begin** property.

BeginGroup property

See also Example

Determines whether the menu item or toolbar item appears at the beginning of a group of items on the menu or toolbar.

Version added

2002

Syntax

boolRet = *object*.**BeginGroup**

object.**BeginGroup** = *boolVal*

boolRet	**Boolean**. **True** if the menu item or toolbar item is at the beginning of a group; otherwise, **False**.
object	Required. An expression that returns a **MenuItem** or **ToolbarItem** object.
boolVal	Required **Boolean**. **True** to indicate that the menu item or toolbar item is the beginning of a group of items; otherwise, **False**.

Remarks

If you set the **BeginGroup** property of a **MenuItem** or **ToolbarItem** object to **True**, a separator is inserted into the menu or a spacer is inserted into the toolbar preceding this item.

Note In Visio 2000, the only way to create a separator in a menu or a spacer in a toolbar was to add a dummy item with a **CmdNum** property of zero, a **Caption** property that contained "", and an empty **MenuItems** or **ToolbarItems** collection. This technique continues to work in Microsoft Visio 2002.

Blue property

Example

Gets or sets the intensity of the blue component of a **Color** object.

Version added

4.0

Syntax

intRet = *object*.**Blue**

object.**Blue** = *intVal*

intRet	**Integer**. The current value of the color's blue component.
object	Required. An expression that returns a **Color** object.
intVal	Required **Integer**. The new value of the color's blue component.

Remarks

The **Blue** property can be a value from 0 to 255.

A color is represented by red, green, and blue components. It also has a flag that indicates how the color is to be used. These correspond to members of the Microsoft Windows **PALETTEENTRY** data structure. For details, search for "**PALETTEENTRY**" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

BottomMargin property

Example

Specifies the bottom margin when printing the pages in a document.

Version added

4.0

Syntax

retVal = object.**BottomMargin**([*unitsNameorCode*])

object.**BottomMargin**([*unitsNameorCode*]) = *newValue*

retVal	**Double**. The margin value expressed in the given units.
object	Required. An expression that returns a **Document** object.
unitsNameorCode	Optional **Variant**. The units to use when retrieving or setting the margin value. Defaults to internal drawing units.
newValue	Required **Double**. The new margin value.

Remarks

The value of this property corresponds to the value entered in the **Bottom** box in

the **Print Setup** dialog box (on the **File** menu, click **Page Setup**, and then click **Setup** on the **Print Setup** tab).

You can specify *unitsNameOrCode* as an integer or a string value. If the string is invalid, an error is generated. For example, the following statements all set *unitsNameOrCode* to inches.

Cell.BottomMargin(**visInches**) = *newValue*

Cell.BottomMargin (65) = *newValue*

Cell.BottomMargin ("in") = *newValue* where "in" can also be any of the alternate strings representing inches, such as "inch", "in.", or "i".

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

Build property

Example

Returns the build number of the running instance.

Version added

2002

Syntax

retVal = object.**Build**

retVal	**Long**. The build number.
object	Required. An expression that returns an **Application** object.

Remarks

The format of the build number is described in the following table.

Bits	Description
0 - 15	Internal build number

The build number of the running instance is written to the

BuildNumberCreated property when a new document is created, and to the **BuildNumberEdited** property when a document is edited.

BuildNumberCreated property

Example

Returns the build number of the instance used to create the document.

Version added

2002

Syntax

retVal = object.**BuildNumberCreated**

retVal	**Long**. The build number when the document was created.
object	Required. An expression that returns a **Document** object.

Remarks

The format of the build number is described in the following table.

Bits	Description
0 - 15	Internal build number

BuildNumberEdited property

Example

Returns the build number of the instance last used to edit the document.

Version added

2002

Syntax

retVal = object.**BuildNumberEdited**

retVal	**Long**. The build number when the document was last edited.
object	Required. An expression that returns a **Document** object.

Remarks

The format of the build number is described in the following table.

Bits	Description
0 - 15	Internal build number

BuiltIn property

See also Example

Determines whether an object is a default Microsoft Visio user interface object or a custom object.

Version added

2000

Syntax

boolVal = object.**Builtin**

boolVal	**Boolean**. **True** if the object is built-in; **False** if it isn't.
object	Required. An expression that returns an object in the **Applies to** list.

BuiltInMenus property

Returns a **UIObject** object that represents a copy of the built-in Microsoft Visio menus and accelerators.

Version added

4.0

Syntax

objRet = *object*.**BuiltInMenus**

objRet	A **UIObject** object that represents the built-in Visio menus and accelerators.
object	Required. An expression that returns an **Application** object.

Remarks

You can use the **BuiltInMenus** property to obtain a **UIObject** object and modify its menus and accelerators. You can then use the **SetCustomMenus** method of an **Application** or **Document** object to substitute your customized menus and accelerators for the built-in Visio menus and accelerators.

You can also use the **SaveToFile** method of the **UIObject** object to store its menus in a file and reload them as custom menus by setting the **CustomMenusFile** property of an **Application** or **Document** object.

BuiltInToolbars property

Returns a **UIObject** object that represents a copy of the built-in Microsoft Visio toolbars.

Version added

4.0

Syntax

objRet = *object*.**BuiltInToolbars**

objRet	A **UIObject** object that represents the built-in Visio toolbars.
object	Required. An expression that returns an **Application** object.

Remarks

You can use the **BuiltInToolbars** property to obtain a **UIObject** object and modify its toolbars. You can then use the **SetCustomToolbars** method of an **Application** or **Document** object to substitute your customized toolbars for the built-in Visio toolbars.

You can also use the **SaveToFile** method of the **UIObject** object to store its toolbars in a file and reload them as custom toolbars by setting the **CustomToolbarsFile** property of an **Application** or **Document** object.

Prior to Visio 5.0, the argument for this property (*fWhichToolbars)* designated which type of toolbar to get (MSOffice or LotusSS). Beginning with Visio 5.0, the application no longer supports different types of toolbars and this argument is ignored.

Caption property

Gets or sets the caption for an object.

Version added

4.0

Syntax

object.**Caption** = *stringVal*

stringVal = *object*.**Caption**

object	Required. An expression that returns an object in the **Applies to** list that has or gets the caption.
stringVal	Required **String**. The caption of the object.

- Use & in the string to cause the next character in the string to become the shortcut key for that menu or menu item, For example, the string "F**&o**rmat" causes **o** to become the shortcut key for that menu item in that one menu.
- Use "" in the string to display a double quotation mark on the

menu.

- Use && in the string to display an ampersand on the menu.

Remarks

Visio does not use the **Caption** property of a **MenuSet** or **ToolbarSet** object.

Category property

Example

Gets or sets the value of a document's category—one of the document properties.

Version added

5.0

Syntax

strRet = *object*.**Category**

object.**Category** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The new value of the field.

Remarks

Setting the **Category** property is equivalent to entering information in the **Category** box in the **Properties** dialog box (click **Properties** on the **File** menu).

Cell[U] property

See also Example

Returns a cell using the name or index of the cell.

Version added

2000

Syntax

objRet = *object*.**Cell**(*reference*)

objRet	A **Cell** object.
object	Required. An expression that returns a **Row** object.
reference	Required **Variant**. The name or index of the cell.

Remarks

The first cell in a row has an index of zero (0).

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **Cell** property to get a **Cell** object using its local name. Use the **CellU** property to get a **Cell** object using its universal name.

CellExists[U] property

Example

Determines whether a particular ShapeSheet cell exists in the scope of the search.

Version added

4.0

Syntax

intRet = object.**CellExists**(*stringExpression*, *fExistsLocall*

intRet	**Integer**. **False** (0) if cell doesn't exist; **True** (-1) if it does.
stringExpression	Required **String**. The name of the ShapeSheet cell for which yc want to search.
fExistsLocally	Required **Integer**. The scope of the search.

Remarks

The *stringExpression* argument must specify a cell name. To search for a cell by section, row, and column index, use the **CellsSRCExists** property.

The *fExistsLocally* argument specifies the scope of the search.

If *fExistsLocally* is non-zero (**True**), the **CellExists** property value is **True** only if the object contains the cell locally; if the cell is inherited, the **CellExists** property value is **False**.

If *fExistsLocally* is zero (**False**), the **CellExists** property value is **True** if the object either contains or inherits the cell.

For a list of cell index values, view the Visio type library for the members of class **VisCellIndices**.

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **CellExists** property to determine if a cell exists using the cell's local name. Use the **CellExistsU** property to determine if a cell exists using the cell's universal name.

Cells[U] property

Returns a **Cell** object that represents a ShapeSheet cell.

Version added

2.0

Syntax

objRet = object.**Cells** (*stringExpression*)

objRet	A **Cell** object that represents the requested cell.
object	Required. An expression that returns a **Shape** or **Style** object that owns the cell.
stringExpression	Required **String**. The name of a ShapeSheet cell.

Remarks

Cells("somestring") does not raise an exception if "somestring" does not name an actual cell. Subsequent methods invoked on the returned object will fail. You can determine if a cell with the name "somestring" exists using the **CellExists** property.

The cells in a shape's User-Defined Cells and Custom Properties sections belong to rows whose names have been assigned by the user or a program. You can access cells in named rows using the **Cells** property.

For example, if "MyRowsName" is the name of a row in a shape's User-Defined Cells section, you can access the zero'th (value) cell in this row using this statement:

```
cellobj = shpobj.cells("User.MyRowsName")
```

You can access the prompt cell in MyRowsName using this statement:

```
cellobj = shpobj.cells("User.MyRowsName.Prompt")
```

Next, assume that MyRowsName is in the Custom Properties section instead of the User- Defined Cells section. You can access the zero'th (value) cell using this statement:

```
cellobj = shpobj.cells("Prop.MyRowsName")
```

You can access other cells in the row using this statement:

```
cellobj = shpobj.cells("Prop.MyRowsName.xxx")
```

where xxx is one of these properties: **Label**, **Prompt**, **SortKey**, **Type**, **Format**, **Invisible**, or **Ask**.

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **Cells** property to get a **Cell** object using the cell's local name. Use the **CellsU** property to get a **Cell** object using the cell's universal name.

CellsC property

See also Example

Returns a **Cell** object that represents a ShapeSheet cell in a layer.

Version added

4.0

Syntax

objRet = *object*.**CellsC**(*column*)

objRet	A **Cell** object that represents the requested cell.
object	Required. An expression that returns a **Layer** object.
column	Required **Integer**. The cell index of the cell to get.

Remarks

The following constants for the cell index are declared by the Visio type library.

Constant	Value
visLayerName	0
visLayerColor	2
visLayerStatus	3
visLayerVisible	4
visLayerPrint	5
visLayerActive	6
visLayerLock	7

visLayerSnap	8
visLayerGlue	9
visLayerNameUniv	10

CellsRowIndex[U] property

See also Example

Returns the index of a row to which a cell belongs.

Version added

2000

Syntax

intRet = *object*.**CellsRowIndex**(*stringExpression*)

intRet	Optional **Integer**. The index of the row containing the cell named in *stringEspression*.
object	Required. An expression that returns a **Shape** object that contains the cell.
stringExpression	Required **String**. The name of a ShapeSheet cell.

Remarks

Beginning with Visio 2000 products, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **CellsRowIndex** property to get a cell's row index using the cell's local name. Use the **CellsRowIndexU** property to get a cell's row index using the cell's universal name.

CellsSRC property

Returns a **Cell** object that represents a ShapeSheet cell identified by section, row, and column indices.

Version added

2.0

Syntax

objRet = *object*.**CellsSRC** (*section*, *row*, *column*)

objRet	A **Cell** object that represents the requested cell.
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The cell's section index.
row	Required **Integer**. The cell's row index.
column	Required **Integer**. The cell's column index.

Remarks

To access any shape formula by its section, row, and column indices, use the

CellsSRC property. Constants for section, row, and column indices are declared by the Visio type library as members of **VisSectionIndices**, **VisRowIndices**, and **VisCellIndices**, respectively.

The **CellsSRC** property does not raise an exception if index values for section, row, and column do not identify an actual cell. Subsequent methods invoked on the returned object fail. You can determine if a cell with particular index values exists using the **CellsSRCExists** property.

The **CellsSRC** property is typically used to iterate through the cells in a section or row. To retrieve a single cell, use the **Cells** property and specify a cell name. For example:

```
Set celObj = Cells("PinX")
```

CellsSRCExists property

Example

Determines whether a ShapeSheet cell exists in the scope of a search.

Version added

4.0

Syntax

intRet = object.**CellsSRCExists**(*section*, *row*, *column*, *fEx*

intRet	**Integer**. **False** (0) if a cell doesn't exist; **True** (-1) if it does.
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The cell's section index.
row	Required **Integer**. The cell's row index.
column	Required **Integer**. The cell's column index.
fExistsLocally	Required **Integer**. The scope of the search.

Remarks

Constants for section, row, and column indices are declared by the Visio type

library as members of **VisSectionIndices**, **VisRowIndices**, and **VisCellIndices**, respectively.

The *fExistsLocally* argument specifies to the scope of the search:

If *fExistsLocally* is non-zero (**True**), the **CellsSRCExists** property returns **True** only if the object contains the cell locally; if the cell is inherited, the **CellsSRCExists** property returns **False**.

If *fExistsLocally* is zero (**False**), the **CellsSRCExists** property returns **True** if the object either contains or inherits the cell.

To search for a cell by name, use the **CellExists** property.

Characters property

Returns a **Characters** object that represents the text of a shape.

Version added

3.0

Syntax

objRet = *object*.**Characters**

objRet	A **Characters** object that represents the shape's text.
object	Required. An expression that returns a **Shape** object.

CharCount property

Example

Returns the number of characters in an object.

Version added

3.0

Syntax

intRet = object.**CharCount**

intRet	**Integer**. The number of characters in the object's text.
object	Required. An expression that returns the **Characters** or **Shape** object that contains the text.

Remarks

For a **Shape** object, the **CharCount** property returns the number of characters in the shape's text. For a **Characters** object, the **CharCount** property returns the number of characters in the text range represented by that object.

The value returned by the **CharCount** property includes the expanded number

of characters for any fields in the object's text. For example, if the text contains a field that displays the file name of a drawing, the **CharCount** property includes the number of characters in the file name, rather than the one-character escape sequence used to represent a field in the **Text** property of a **Shape** object.

CharProps property

Example

Sets a character property of a **Characters** object to a new value.

Version added

3.0

Syntax

object.**CharProps**(*intWhichProp*) = *intExpression*

object	Required. An expression that returns a **Characters** object.
intWhichProp	Required **Integer**. The property to set.
intExpression	Required **Integer**. The new value for the property.

Remarks

Depending on the extent of the text range and the format, setting the **CharProps** property may cause rows to be added or removed from a shape's Character section.

The **CharProps** property is a write-only property. To retrieve formatting

properties of a **Characters** object, use the **CharPropsRow** property.

The values of the *intWhichProp* argument correspond to cells viewed in the Character section of the ShapeSheet window, and the values of *intExpression* correspond to the values that can be entered in those cells in the ShapeSheet window.

Constants for *intWhichProp* and *intExpression* are declared by the Visio type library.

intWhichProp	**Value**	***intExpression***	**Value**
visCharacterFont	0	An integer that represents an index into the fonts collection installed on a system. Zero represents the default font.	N/A
visCharacterColor	1	An integer from 0 to 23 that corresponds to a color in the current color palette.	N/A
visCharacterStyle	2	**visBold** **visItalic** **visUnderLine** **visSmallCaps**	&H1 &H2 &H4 &H8
visCharacterCase	3	**visCaseNormal** **visCaseAllCaps** **visCaseInitialCaps**	0 1 2
visCharacterPos	4	**visPosNormal** **visPosSuper** **visPosSub**	0 1 2
visCharacterSize	7	An integer representing point size.	N/A

CharPropsRow property

Example

Returns the index of the row in the Character section of a ShapeSheet window that contains character formatting information for a **Characters** object.

Version added

3.0

Syntax

intRet = *object*.**CharPropsRow**(*bias*)

intRet	**Integer**. The index of the row that defines the **Characters** object's format.
object	Required. An expression that returns a **Characters** object.
bias	Required **Integer**. The direction of the search.

Remarks

If the formatting of the **Characters** object is represented by more than one row in the Character section of the ShapeSheet window, the **CharPropsRow** property returns -1. If the **Characters** object represents an insertion point rather

than a sequence of characters (that is, if its **Begin** and **End** properties return the same value), use the *bias* argument to determine which row index to return.

Constant	Value
visBiasLeft	1
visBiasRight	2
visBiasLetVisioChoose	0

Specify **visBiasLeft** for the row that covers character formatting for the character to the left of the insertion point, or **visBiasRight** for the row that covers character formatting for the character to the right of the insertion point.

CharSet property

Example

Returns the Microsoft Windows character set for a **Font** object.

Version added

4.0

Syntax

intRet = *object*.**CharSet**

intRet	**Integer**. The character set code for the object.
object	Required. An expression that returns a **Font** object.

Remarks

The Windows character set specifies character mapping for a font. The possible values of the **CharSet** property correspond to those of the **lfCharSet** member of the Windows **LOGFONT** data structure. For details, search for "**LOGFONT**" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

ClassID property

Returns the class ID string of a shape representing an ActiveX control or an embedded or linked OLE object.

Version added

4.5

Syntax

strRet = *object*.**ClassID**

strRet	**String**. The class ID of the OLE object represented by the shape.
object	Required. An expression that returns the **Shape** object to examine.

Remarks

The **ClassID** property raises an exception if the shape doesn't represent an ActiveX control or OLE 2.0 embedded or linked object. A shape represents an ActiveX control or an OLE 2.0 embedded or linked object if the

visTypeIsOLE2 bit (&H8000) is set in the value returned by *shpObj*.**ForeignType**.

ClassID returns a string of the form:

{2287DC42-B167-11CE-88E9-002AFDDD917}

This identifies the application that services the object. It might, for example, identify an embedded object on a Visio page as a Microsoft Excel object.

After using a shape's **Object** property to obtain an Automation interface on the object the shape represents, you might want to obtain the shape's **ClassID** or **ProgID** property to determine the methods and properties provided by the interface.

Clone property

See also

Returns a copy of the **UIObject** object.

Version added

2000

Syntax

objRet = *object*.**Clone**

objRet	A copy of the **UIObject** object.
object	Required. An expression that returns a **UIObject** object.

Closed property

See also Example

Determines if the object is closed (its begin point coincides with its end point).

Version added

5.0

Syntax

intRet = *object*.**Closed**

intRet	**Integer**. **True** (-1) if the **Path** or **Curve** object is closed; otherwise, **False** (0).
object	Required. An expression that returns a **Path** or **Curve** object to examine.

Remarks

Use the **Closed** property of a **Path** or **Curve** object to test for equality (Visio uses 10E-6 as its "fuzz" factor) of the object's begin and end points. A closed **Curve** object can be in a **Path** object that is open, and a **Curve** object that is open can be in a closed **Path** object.

The **Closed** property of a **Path** object is unrelated to a **Path** object's fill. A **Path** object is filled if its Geometry*n*.NoFill cell is zero (0). If Visio is told to fill an open **Path** object, it pretends there is a LineTo cell from the **Path** object's end point to its begin point. When filling a **Path** object, Visio considers a point to be inside the **Path** object if a ray drawn from the point in any direction crosses the

Path object or any of the shape's other **Path** objects cross an odd number of times.

CmdNum property

Gets or sets the command ID associated with an accelerator, menu, menu item, or toolbar item.

Version added

4.0

Syntax

object.**CmdNum** = *intVal*

intVal = *object*.**CmdNum**

object	Required. An expression that returns an **AccelItem**, **Menu**, **MenuItem**, or **ToolbarItem** object.
intVal	Required **Integer**. The command ID of the object.

Remarks

When the **AddOnName** property of a **MenuItem** or **ToolbarItem** object indicates an add-on to run, Visio automatically assigns a **CmdNum** property.

The **CmdNum** property for a **MenuItem** object that represents a submenu should be zero (0). The **CmdNum** property should never be zero for an **AccelItem** object.

To insert a separator in a menu preceding a **MenuItem** object or a spacer in a toolbar preceding a **ToolbarItem** object, use the **BeginGroup** property.

Valid command IDs are declared by the Visio type library in **VisUICmds**. They have the prefix **visCmd**.

CntrlID property

See also Example Applies to

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions of Visio, this property contained the control ID for a menu, menu item, or toolbar item.

CntrlType property

Gets or sets the control type of a menu, menu item, or toolbar item.

Version added

4.0

Syntax

object.**CntrlType** = *intVal*

intVal = *object*.**CntrlType**

object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** object.
intVal	Required **Integer**. The control type of the object.

Remarks

If you are adding a custom toolbar button, set the **CntrlType** property to **visCtrlTypeBUTTON**. The following table describes the control types declared by the Visio type library in **VisUICtrlTypes**.

Constant	Value
visCtrlTypeBUTTON	2 (&H2)
visCtrlTypeSPLITBUTTON	16 (&H10)
visCtrlTypeSPLITBUTTON_MRU_COLOR	17 (&H11)
visCtrlTypeSPLITBUTTON_MRU_COMMAND	18 (&H12)
visCtrlTypeBUTTON_OWNERDRAW	33 (&H21)
visCtrlTypeEDITBOX	64 (&H40)
visCtrlTypeCOMBOBOX	128 (&H80)
visCtrlTypeCOMBOBOX_SORTED	129 (&H81)
visCtrlTypeDROPDOWN_OWNERDRAW	256 (&H100)
visCtrlTypeDROPDOWN_SORTED_OWNERDRAW	257 (&H101)
visCtrlTypeDROPDOWN	272 (&H110)
visCtrlTypeDROPDOWN_SORTED	273 (&H111)
visCtrlTypeLABEL	2048 (&H800)
visCtrlTypeSWATCH	32768 (&H8000)
visCtrlTypeSWATCH_COLORS	32769 (&H8001)

Colors property

Example

Returns the **Colors** collection of a **Document** object.

Version added

4.0

Syntax

objRet = *object*.**Colors**

objRet	The **Colors** collection of the **Document** object.
object	Required. An expression that returns a **Document** object.

Column property

Example

Returns the column index of a cell.

Version added

4.0

Syntax

intRet = *object*.**Column**

intRet	**Integer**. The column index of the **Cell** object.
object	Required. An expression that returns a **Cell** object.

COMAddIns property

See also

Returns a reference to the **COMAddIns** collection that represents all the Component Object Model (COM) add-ins currently registered in Microsoft Visio.

Version added

2002

Syntax

objsRet = *object*.**COMAddIns**

objsRet	A collection of **COMAddIn** objects that provide information about a COM add-in registered in the registry.
object	Required. An expression that returns an **Application** object.

Remarks

The COM add-ins that are currently registered are listed in the **COM Add-Ins** dialog box (on the **Tools** menu, point to **Macros**, and then click **COM Add-ins**).

To get information about the object returned by the **COMAddIns** property:

On the **Tools** menu, point to **Macros**, and then click **Visual Basic Editor**.

On the **View** menu, click **Object Browser**.

In the **Project/Library** list, click **Office**.

> If you do not see the Office type library in the **Project/Library** list, on the **Tools** menu, click **References**, select the **Microsoft Office 10.0 Object Library** check box, and then click **OK**.

Under **Classes**, examine the class named **COMAddIns**.

Example

```
'This macro demonstrates using the COMAddIns property
'to list the COM add-ins registered with Visio.
Public Sub IterateCOMAddIns()

    Dim myCOMAddIns As COMAddIns
    Dim myCOMAddIn As COMAddIn

    'Get the set of COM add-ins
    Set myCOMAddIns = Application.COMAddIns

    'List each COM add-in in the
    'Immediate window
    For Each myCOMAddIn In myCOMAddIns
       Debug.Print myCOMAddIn.Description
    Next
```

```
End Sub
```

CommandBars property

Returns a reference to the **CommandBars** collection that represents the command bars in the container application.

Version added

2002

Syntax

objsRet = *object*.**CommandBars**

objsRet	The collection of **CommandBar** objects that represent command bars in the container application.
object	Required. An expression that returns an **Application** object.

Remarks

Beginning with Microsoft Visio 2002, a program can manipulate menus and toolbars in the Visio user interface by manipulating the **CommandBars** collection returned by the **CommandBars** property. The **CommandBars** collection has an interface identical to the **CommandBars** collection exposed by

the suite of Microsoft Office applications such as Microsoft Word and Microsoft Excel.

Alternatively, since Visio version 4.0, Visio has exposed application and document properties that return a **UIObject** object that provides similar functionality to **CommandBars**. Consequently, programs can manipulate the Visio menus and toolbars using either the **CommandBars** collection or **UIObject** objects.

To get information about the object returned by the **CommandBars** property:

On the **Tools** menu, point to **Macros**, and then click **Visual Basic Editor**.

On the **View** menu, click **Object Browser**.

In the **Project/Library** list, click **Office**.

> If you do not see the Office type library in the **Project/Library** list, on the **Tools** menu, click **References**, select the **Microsoft Office 10.0 Object Library** check box, and then click **OK**.

Under **Classes**, examine the class named **CommandBars**.

Note Each **CommandBarControl** object in a **CommandBars** collection has an **OnAction** and **Context** property whose values are determined by the container application. In Visio:

The **OnAction** property is a **String** value that is interpreted either as a COM add-in, a Visual Basic for Applications (VBA) macro, VBA code or as a Visio add-on name.

The **Context** property determines in which menu context a command bar appears. The menu context number is a **String** value (for example **visUIObjSetDrawing** or "2"), which is followed by an asterisk if the command bar is visible by default (for example, **visUIObjSetShapeSheet** & "*" or "4*"). Valid menu contexts are **visUIObjSetDrawing** (2), **visUIObjSetStencil** (3), **visUIObjSetShapeSheet** (4), **visUIObjSetIcon**(5), or **visUIObjSetPrintPreview** (7). Attempting to set the **Context** property to any other value will fail.

For more information about using the **OnAction** and **Context** properties in Visio, see Developing Visio Solutions on the Microsoft Developer Network (MSDN) Web site.

Example

```
'This macro demonstrates using the CommandBars proper
'to list the command bars.

Public Sub IterateCommandBars()

    Dim myCommandBars As CommandBars
    Dim myCommandBar As CommandBar

    'Get the set of CommandBars
    'for the application
    Set myCommandBars = Application.CommandBars

    'List each CommandBar in
    'the Immediate window
    For Each myCommandBar In myCommandBars
        Debug.Print myCommandBar.Name
    Next

End Sub
```

CommandLine property

See also Example

Determines how Microsoft Visio was started.

Version added

2000

Syntax

strRet = *object*.**CommandLine**

strRet	**String**. The command line with which the application was started.
object	Required. An expression that returns an **Application** object.

Remarks

When you double-click a drawing, template, or stencil icon to start the application, the name of the file appears in the string returned by the **CommandLine** property. When you start the application using a **CreateObject** call, "/Automation" appears in the string. When you double-click a Visio embedded object in an OLE container application, "/Embedding" appears in the string.

The following table includes other command line switches you can use to start the application.

Command line switch	Description

/nonew	**Choose Drawing Type** dialog box is not shown on startup.
/nologo	Splash screen is not shown on startup.
/p filename	**Print** dialog box is shown so you can quickly print a file.
filename	Opens a Visio file. The file has to be in the **Drawings** file path on the **File Paths** tab in the **Options** dialog box (on the **Tools** menu, click **Options**), or an absolute path must be named.
/1, /2, /3,…/9	Opens one of the last-opened files.
/noreg	Prevents Visio from registering itself.
/u	Unregisters Visio.
/r	Registers Visio.
/s	Silently registers Visio.
/pt filename, [printername, drivername, portname]	Directs file to print on a particular printer. (Added in Visio version 5.0c.)
::ODMA	Visio opens a file using ODMA.

Company property

Example

Returns or sets the value of the **Company** field in a document's properties.

Version added

5.0

Syntax

strRet = *object*.**Company**

object.**Company** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The new value of the field.

Remarks

Setting the **Company** property is equivalent to entering information in the **Company** box in the **Properties** dialog box (click **Properties** on the **File** menu).

Connects property

Example

Returns a **Connects** collection for a shape, page, or master.

Version added

2.0

Syntax

objRet = *object*.**Connects**

objRet	The **Connects** collection of the **Shape**, **Page**, or **Master** object.
object	Required. An expression that returns a **Shape**, **Page**, or **Master** object that owns the collection.

Remarks

The **Connects** collection of a shape contains every **Connect** object for which the shape is returned by the **FromSheet** property. This tells you all the shapes to which the shape is connected.

To obtain a **Connects** collection that contains every **Connect** object for which

the shape is the **ToSheet** property, use the shape's **FromConnects** property. This tells you all the shapes that are connected to this shape.

The **Connects** collection of a page contains a **Connect** object for every connection on the page.

The **Connects** collection of a master contains a **Connect** object for every connection in the master.

Container property

See also Example

Returns an **IDispatch** interface on the ActiveX container in which the document is contained or **Nothing** if the document is not in a container.

Version added

2000

Syntax

objRet = *object*.**Container**

objRet	An **IDispatch** on the container.
object	Required. An expression that returns a **Document** object.

Remarks

The interface returned is the result of querying the **IOleContainer** interface provided by the containing object for **IDispatch**.

ContainingMaster property

Example

Returns the **Master** object that contains an object.

Version added

4.0

Syntax

objRet = object.**ContainingMaster**

objRet	The **Master** object that contains the object or collection.
object	Required. An expression that returns a **Selection** or **Shape** object, or **Shapes** collection to examine.

Remarks

If the object isn't in a **Master** object, the **ContainingMaster** property returns **Nothing**. For example, if a **Shape** object belongs to the **Shapes** collection of a **Page** object, the **ContainingMaster** property returns **Nothing**.

ContainingPage property

Example

Returns the page that contains an object.

Version added

4.0

Syntax

objRet = object.**ContainingPage**

objRet	The **Page** object that contains the object or collection.
object	Required. An expression that returns a **Selection** or **Shape** object or a **Shapes** collection to examine.

Remarks

If the object isn't in a **Page** object, the **ContainingPage** property returns **Nothing**. For example, if a **Shape** object belongs to a **Masters** collection, the **ContainingPage** property returns **Nothing**.

ContainingRow property

See also Example

Returns the row that contains a cell.

Version added

2000

Syntax

objRet = *object*.**ContainingRow**

objRet	The **Row** object that contains the **Cell** object.
object	Required. An expression that returns a **Cell** object.

ContainingSection property

See also Example

Returns the section in which a row is contained.

Version added

2000

Syntax

objRet = *object*.**ContainingSection**

objRet	The **Section** object that contains the row.
object	Required. An expression that returns a **Row** object.

ContainingShape property

Example

Returns the **Shape** object that contains an object or collection.

Version added

4.0

Syntax

objRet = object.**ContainingShape**

objRet	The **Shape** object that contains the object or collection.
object	Required. An expression that returns a **Selection** or **Shape** object or a **Shapes** collection to examine.

Remarks

If the **Shape** object is the member of a group, the **ContainingShape** property returns that group.

If the **Shape** object is a top-level shape in its **Page** or **Master** object (it is not a member of a group), the **ContainingShape** property returns the page sheet of its

page or master.

If the **Shape** object is the page sheet of a page or master, the **ContainingShape** property returns **Nothing**.

ContainsWorkspace property

Example

Determines whether the document was saved as a workspace.

Version added

2002

Syntax

boolRet = object.**ContainsWorkspace**

boolRet	**Boolean**. True if the document was saved as a workspace; otherwise, **False**.
object	Required. An expression that returns a **Document** object.

Remarks

This property is read only, but you can cause a document to be saved as a workspace by passing the **visSaveAsWS** flag to the **SaveAsEx** method.

This is equivalent to clicking **Workspace** under the **Save** button in the **Save As** dialog box (on the **File** menu, click **Save As**, and then click the arrow next to

Save).

Control property

Example

Determines whether the CTRL key modifies the accelerator key in an **AccelItem** object.

Version added

3.0

Syntax

intRet = *object*.**Control**

***object*.Control** = *intExpression*

intRet	**Integer**. **True** (-1) if the CTRL key modifies the key in an **AccelItem** object; otherwise, **False** (0).
object	Required. An expression that returns an **AccelItem** object.
intExpression	Required **Integer**. **True** (non-zero) if the CTRL key modifies the key in an **AccelItem** object; otherwise, **False** (0).

Remarks

Set the **Control** property to **True** to use the CTRL key as a modifier for an accelerator, for example, CTRL+BACKSPACE.

Count property

See also

Returns the number of objects in a collection.

Version added

2.0

Syntax

intRet = *object*.**Count**

intRet	**Integer**. The number of objects in the collection.
object	Required. An expression that returns an object from the **Applies to** list.

Creator property

Gets or sets the value of a document's author.

Version added

2.0

Syntax

strRet = *object*.**Creator**

object.**Creator** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The new value of the field.

Remarks

Setting the **Creator** property is equivalent to entering information in the **Author** box in the **Properties** dialog box (click **Properties** on the **File** menu).

CurrentScope property

Determines the ID of the scope that causes an event to fire.

Version added

2000

Syntax

longRet = object.**CurrentScope**

longRet	**Long**. The scope ID of the most recently fired **EnterScope** event; **visScopeIDInvalid** (-1) if a scope isn't open.
object	Required. An expression that returns an **Application** object.

Remarks

The scope ID could be an internal Visio scope ID that corresponds to a Visio command or an external scope ID passed to an Automation client through the **BeginUndoScope** method.

The recipients of an event consider a scope open if the **EnterScope** event has

fired, but the **ExitScope** event has not fired.

To determine if the event queue firing is related to a particular scope internal to the application or one opened and closed by an Automation client, use the **IsInScope** property.

CustomMenus property

Gets a **UIObject** object that represents the current custom menus and accelerators of an **Application** object or a **Document** object.

Version added

4.0

Syntax

objRet = *object*.**CustomMenus**

objRet	A **UIObject** object that represents the object's current custom menus.
object	Required. An expression that returns the **Application** or **Document** object to examine.

Remarks

Beginning with Visio 2000, if you haven't customized the user interface, the object returned by **CustomMenus** property contains the same collections as the object returned by the **BuiltInMenus** property.

In Visio 5.0 or earlier, if the object is not using custom toolbars, the **CustomMenus** property returns **Nothing**.

CustomMenusFile property

Gets or sets the name of the file that defines custom menus and accelerators for an **Application** object or a **Document** object.

Version added

4.0

Syntax

strRet = *object*.**CustomMenusFile**

object.**CustomMenusFile** = *fileStr*

strRet	**String**. The name of the file that defines the current custom menus for the object.
object	Required. An expression that returns the **Application** or **Document** object.
fileStr	Required **String**. The name of the file that defines new custom menus for the object.

Remarks

If the object is not using custom menus, the **CustomMenusFile** property returns **Nothing**.

CustomToolbars property

Example

Gets a **UIObject** object that represents the current custom toolbars and status bars of an **Application** object or a **Document** object.

Version added

4.0

Syntax

objRet = *object*.**CustomToolbars**

objRet	A **UIObject** object that represents the object's current custom toolbars.
object	Required. An expression that returns an **Application** or **Document** object.

Remarks

Beginning with Visio 2000, if you haven't customized the user interface, the object returned by **CustomToolbars** property contains the same collections as the object returned by the **BuiltInToolbars** property.

In Visio 5.0 or earlier, if the object is not using custom toolbars, the **CustomToolbars** property returns **Nothing**.

CustomToolbarsFile property

Example

Returns or sets the name of the file that defines custom toolbars and status bars for an **Application** object or a **Document** object.

Version added

4.0

Syntax

strRet = *object*.**CustomToolbarsFile**

object.**CustomToolbarsFile** = *fileStr*

strRet	**String**. The name of the file that defines the current custom toolbars for the object.
object	Required. An object that returns an **Application** or **Document** object.
fileStr	Required **String**. The name of the file that defines new custom toolbars for the object.

Remarks

If the object is not using custom toolbars, the **CustomToolbarsFile** property returns **Nothing**.

Data1 property

Gets or sets the value of the **Data1** field for a **Shape** object.

Version added

2.0

Syntax

strRet = *object*.**Data1**

object.**Data1** = *strExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Shape** object.
strExpression	Required **String**. The new value for the field.

Remarks

Use the **Data1** property to supply additional information about a shape. The property can contain up to 64 KB of characters. Text controls should be used with care with a string that is greater than 3,000 characters.

Setting the **Data1** property is equivalent to entering information in the **Data 1** box in the **Special** dialog box (click **Special** on the **Format** menu).

Data2 property

Gets or sets the value of the **Data2** field for a **Shape** object.

Version added

2.0

Syntax

strRet = *object*.**Data2**

object.**Data2** = *strExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Shape** object.
strExpression	Required **String**. The new value for the field.

Remarks

Use the **Data2** property to supply additional information about a shape. The property can contain up to 64 KB of characters. Text controls should be used with care with a string that is greater than 3,000 characters.

Setting the **Data2** property is equivalent to entering information in the **Data 2** box in the **Special** dialog box (click **Special** on the **Format** menu).

Data3 property

Gets or sets the value of the **Data3** field for a **Shape** object.

Version added

2.0

Syntax

strRet = *object*.**Data3**

object.**Data3** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Shape** object.
stringExpression	Required **String**. The new value for the field.

Remarks

Use the **Data3** property to supply additional information about a shape. The property can contain up to 64 KB of characters. Text controls should be used with care with a string that is greater than 3,000 characters.

Setting the **Data3** property is equivalent to entering information in the **Data 3** box in the **Special** dialog box (click **Special** on the **Format** menu).

DefaultAngleUnits property

Example

Determines the default unit of measure for quantities that represent angles.

Version added

2002

Syntax

unitsCode = *object*.**DefaultAngleUnits**

object.**DefaultAngleUnits** = *unitsNameOrCode*

unitsCode	**Variant**. The default angle unit.
object	Required. An expression that returns an **Application** object.
unitsNameOrCode	Optional **Variant**. The new default angle unit.

Remarks

The **DefaultAngleUnits** property corresponds to the value shown in the **Angle** box on the **Regional** tab in the **Options** dialog box (on the **Tools** menu, click **Options**).

The return value *unitsCode* contains one of the values of **VisUnitCodes**, which are declared in the Visio type library.

You can specify *unitsNameOrCode* as an integer (a member of **VisUnitCodes**) or a string value such as "degrees". If the string is invalid or the unit code is inappropriate (non-angular), an error is generated.

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Cell formulas that contain a specific unit of measure are displayed in those units regardless of the default angle units setting. Many cell formulas, however, use implicit unit syntax and are displayed in default units.

A program can create a cell whose formula is displayed in default units by setting the cell's **Formula** property to a string in implicit unit syntax. For example, if the formula for the angle of a shape is "=90[deg,A]" , the result is displayed as "90 deg." if the **DefaultAngleUnits** property is **visDegrees**, and "1.5708 rad." if the **DefaultAngleUnits** property is **visRadians**.

Alternatively, a program can set the cell's result to default angle units using the following statement:

cellObj.**Result(visAngleUnits)** = 90

In this case, the result is 90 degrees if the **DefaultAngleUnits** property is **visDegrees**, and 90 radians if the **DefaultAngleUnits** property is **visRadians**.

For details about implicit units of measure, see About units of measure.

DefaultDurationUnits property

Example

Determines the default unit of measure for quantities that represent durations.

Version added

2002

Syntax

unitsCode = *object*.**DefaultDurationUnits**

object.**DefaultDurationUnits** = *unitsNameOrCode*

unitsCode	**Variant**. The default duration unit of measure.
object	Required. An expression that returns an **Application** object.
unitsNameOrCode	Optional **Variant**. The new default duration unit of measure.

Remarks

The **DefaultDurationUnits** property corresponds to the value shown in the **Duration** box on the **Regional** tab in the **Options** dialog box (on the **Tools** menu, click **Options**).

The return value *unitsCode* contains one of the values of **VisUnitCodes**, which are declared in the Visio type library.

You can specify *unitsNameOrCode* as an integer (a member of **VisUnitCodes**) or a string value such as "minutes". If the string is invalid or the unit code is inappropriate (non-duration), an error is generated.

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Cell formulas that contain a specific unit of measure are displayed in those units regardless of the default duration units setting. Many cell formulas, however, use implicit unit syntax and are displayed in default units.

A program can create a cell whose formula displays in default units by setting the cell's **Formula** property to a string in implicit unit syntax. For example, if a formula specifying duration is "=10[em,E]" , the result displays as "0.0069 ed" if the **DefaultDurationUnits** property is **visElapsedDay**, and "600.0000 es" if the **DefaultDurationUnits** property is **visElapsedSec**.

Alternatively, a program can set the cell's result to default duration units using the following statement:

cellObj.**Result(visDurationUnits)** = 60

In this case, the result is 60 minutes if the **DefaultDurationUnits** property is **visElapsedMin** and 60 seconds if the **DefaultDurationUnits** property is **visElapsedSec**.

For details about implicit units of measure, see About units of measure.

DefaultFillStyle property

Example

Gets or sets the default fill style of a document.

Version added

4.0

Syntax

strRet = *object*.**DefaultFillStyle**

object.**DefaultFillStyle** = *stringExpression*

strRet	**String**. The default fill style of the document.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The name of the default fill style to assign to the document.

Remarks

The **DefaultFillStyle** property corresponds to the value shown in the **Fill Style** box on the **Format Shape** toolbar when nothing is selected on the drawing page.

The document's default fill style is applied to new shapes created with the Visio drawing tools or with the **Draw** methods via Automation.

DefaultGuideStyle property

See also Example

Gets or sets the default guide style of a document.

Version added

2002

Syntax

strRet = *object*.**DefaultGuideStyle**

object.**DefaultGuideStyle** = *stringExpression*

strRet	**String**. The default guide style of the document.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The name of the default guide style.

Remarks

The **DefaultGuideStyle** property specifies what style is applied to new guides created in the document.

DefaultLineStyle property

Example

Gets or sets the default line style of a document.

Version added

4.0

Syntax

strRet = *object*.**DefaultLineStyle**

object.**DefaultLineStyle** = *stringExpression*

strRet	**String**. The default line style of the document.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The name of the default line style to assign to the document.

Remarks

The **DefaultLineStyle** property corresponds to the value shown in the **Line Style** box on the **Format Shape** toolbar when nothing is selected on the drawing

page. The document's default line style is applied to new shapes created with the Visio drawing tools or with the **Draw** methods via Automation.

DefaultPageUnits property

Example

Determines the default unit of measure for quantities that represent position or distance.

Version added

2002

Syntax

unitsCode = *object*.**DefaultPageUnits**

object.**DefaultPageUnits** = *unitsNameOrCode*

unitsCode	**Variant**. The default page units.
object	Required. An expression that returns an **Application** object.
unitsNameOrCode	Optional **Variant**. New default page units.

Remarks

The **DefaultPageUnits** property corresponds to the value shown in the **Page** box on the **Regional** tab in the **Options** dialog box (on the **Tools** menu, click

Options).

The return value *unitsCode* contains one of the values of **VisUnitCodes**, which are declared in the Visio type library.

You can specify *unitsNameOrCode* as an integer (a member of **VisUnitCodes**) or a string value such as "inches". If the string is invalid or the unit code is inappropriate (non-distance), an error is generated.

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Cell formulas that contain a specific unit of measure are displayed in those units regardless of the default page units setting. Many cell formulas, however, use implicit unit syntax and are displayed in default units.

A program can create a cell whose formula is displayed in default units by setting the cell's **Formula** property to a string in implicit unit syntax. For example, the formula "=5[in,P]" displays as "5 in." if the **DefaultPageUnits** property is **visInches**, and "12.7 cm" if the **DefaultPageUnits** property is **visCentimeters**.

Alternatively, a program can set the cell's result to default page units using the following statement:

cellObj.**Result(visPageUnits)** = 5

In this case, the result is 5 inches if the **DefaultPageUnits** property is **visInches** and 5 centimeters if the **DefaultPageUnits** property is **visCentimeters**.

For details about implicit units of measure, see About units of measure.

The value of the **DefaultPageUnits** property determines how blank drawings are created. If the value is a metric unit of measure, then the blank drawing is created using metric units. If the value is an imperial unit of measure, the blank drawing is created using imperial units. Default page units do not apply to new drawings created from a template.

Other operations can change the value of the **DefaultPageUnits** property. If you change the scale or measurement units for a particular page using the **Page**

Setup dialog box, Visio changes the default page units for all drawings in the application.

Note Visio maintains internal default unit settings for position and distance *in the drawing* (**visDrawingUnits**) as opposed to those *on the page*. Default drawing units cannot be set explicitly, or queried directly. However, they can be inferred from the ratio of *cellObj*.**Result**(**visDrawingUnits**) to *cellObj*.**Result**(*specificUnit*) where *specificUnit* identifies a known unit of length such as **visInches**.

You can also use implicit syntax to create formulas that display in default drawing units, for example, "=5[i,D]".

If you set the value of **DefaultPageUnits** when the current settings for default page units and default drawing units are the same, both settings are changed.

DefaultStyle property

Example

Gets the default fill style of a document or sets the default fill, line, and text styles of a document.

Version added

4.0

Syntax

strRet = *object*.**DefaultStyle**

object.**DefaultStyle** = *stringExpression*

strRet	**String**. The default fill style of the document.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The name of the default style to assign to the document.

Remarks

A document's **DefaultStyle** property returns the same value as its

DefaultFillStyle property. Setting the **DefaultStyle** property is equivalent to setting the **DefaultFillStyle**, **DefaultLineStyle**, and **DefaultTextStyle** properties individually to the same multiple-attribute style. The fill, line, and text attributes of the document's default style are applied to new shapes created with the Visio drawing tools or with the **Draw** methods via Automation.

DefaultTextStyle property

Example

Gets or sets the default text style of a document.

Version added

4.0

Syntax

strRet = *object*.**DefaultTextStyle**

object.**DefaultTextStyle** = *stringExpression*

strRet	**String**. The default text style of the document.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The name of the default text style to assign to the document.

Remarks

The **DefaultTextStyle** property corresponds to the value shown in the **Text Style** box on the **Format Text** toolbar when nothing is selected on the drawing page.

The document's default text style is applied to new shapes created with the Visio drawing tools or with the **Draw** methods via Automation.

DefaultTextUnits property

Example

Determines the default unit of measure for quantities that represent text metrics.

Version added

2002

Syntax

unitsCode = *object*.**DefaultTextUnits**

object.**DefaultTextUnits** = *unitsNameOrCode*

unitsCode	**Variant**. The default text units.
object	Required. An expression that returns an **Application** object.
unitsNameOrCode	Required **Variant**. New default text units.

Remarks

The **DefaultTextUnits** property corresponds to the value shown in the **Text** box on the **Regional** tab in the **Options** dialog box (on the **Tools** menu, click **Options**).

The return value *unitsCode* contains one of the values of **VisUnitCodes**, which are declared in the Visio type library.

You can specify *unitsNameOrCode* as an integer (a member of **VisUnitCodes**) or a string value such as "pt". If the string is invalid or the unit code is inappropriate (non-textual), an error is generated.

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Cell formulas that contain a specific unit of measure are displayed in those units regardless of the default text units setting. Many cell formulas, however, use implicit unit syntax and are displayed in default units.

A program can create a cell whose formula is displayed in default units by setting the cell's **Formula** property to a string in implicit unit syntax. For example, the formula "=8[pt,T]" displays as "8 pt" if the **DefaultTextUnits** property is **visPoints**, and "0.6272" if the **DefaultTextUnits** property is **visCiceros**.

Alternatively, a program can set the cell's result to default text units using the following statement:

cellObj.**Result(visTextUnits)** = 12

In this case, the text is 12 points if the **DefaultTextUnits** property is **visPoints**, and 12 ciceros if the **DefaultTextUnits** property is **visCiceros**.

For details about implicit units of measure, see About units of measure.

DeferRecalc property

Example

Determines whether the application recalculates cell formulas during a series of actions.

Version added

4.1

Syntax

intRet = *object*.**DeferRecalc**

object.**DeferRecalc** = *intExpression*

intRet	**Integer**. **False** (0) if formulas are recalculated as needed; **True** (-1) if recalculation is deferred.
object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. **False** (0) to recalculate formulas as needed; **True** (non-zero) to defer recalculation.

Remarks

Use the **DeferRecalc** property to improve performance during a series of actions. For example, you can defer formula recalculation while changing the formulas or values of several cells.

If a program neglects to turn recalculation on again after turning it off, Visio turns it on when the user performs an operation.

If you release objects or send a large number of commands to Visio while recalculation is deferred, Visio may at times need to process its queue of pending recalculations. Because of this, use care in setting formulas inside a scope where you want recalculation deferred. Ideally, you should only set formulas when recalculation is turned off.

For example, consider the following Microsoft Visual Basic sequence:

```
visObj.DeferRecalc = True
shpObj.Cells("height").ResultIU = 12
shpObj.Cells("width").ResultIU = 14
visObj.DeferRecalc = False
```

Because Visual Basic makes and releases a temporary **Cell** object in the preceding code, Visio will process its queue at that point.

In the following sequence, Visio will not process the recalculation queue until the program turns recalculation on again (or the user performs some operation).

```
visObj.DeferRecalc = True
Set cellObj1 = shpObj.Cells("Height")
Set cellObj2 = shpObj.Cells("Width")
cellObj1.ResultIU = 12
cellObj1.ResultIU = 14
visObj.DeferRecalc = False
```

Description property

Gets or sets the value of the **Description** box in a **Document** object's properties or a shape's **Hyperlink** object.

Version added

2.0

Syntax

strRet = *object*.**Description**

object.**Description** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Document** or **Hyperlink** object.
stringExpression	Required **String**. The new value for the field.

Remarks

Setting a document's **Description** property is equivalent to entering information

in the **Description** box in the **Properties** dialog box (click **Properties** on the **File** menu).

Setting a hyperlink's **Description** property is equivalent to entering information in the optional **Description** box in the **Hyperlinks** dialog box (click **Hyperlinks** on the **Insert** menu). It is also equivalent to setting the result of the Description cell of the shape's Hyperlink.Row row.

DisplayKeysInTooltips property

Example

Determines whether ToolTip text includes keyboard shortcuts.

Version added

2000

Syntax

boolVal = *object*.**DisplayKeysInTooltips**

object.**DisplayKeysInTooltips** = *boolVal*

boolVal	Required **Boolean**. **True** if ToolTip text shows keyboard shortcuts; **False** if it does not.
object	Required. An expression that returns a **UIObject** object.

Remarks

To show ToolTips, you must set the **DisplayTooltips** property to **True**.

It doesn't matter which **UIObject** object you use when getting or setting this

property. The property affects the entire application, and always affects the appearance of ToolTips in the current visible set of toolbars.

Beginning with Microsoft Visio 2002, this setting corresponds to the **Show shortcut keys in ScreenTips** setting on the **Options** tab in the **Customize** dialog box (on the **Tools** menu, click **Customize**), and is shared between Visio 2002 and all Microsoft Office XP applications.

DisplayTooltips property

Example

Determines whether ToolTips are shown in toolbars.

Version added

2000

Syntax

boolVal = *object*.**DisplayTooltips**

object.**DisplayTooltips** = *boolVal*

boolVal	Required **Boolean**; **True** if ToolTips are shown; **False** if they are not.
object	Required. An expression that returns a **UIObject** object.

Remarks

It doesn't matter which **UIObject** object you use when getting or setting this property. The property affects the entire application, and always affects the appearance of ToolTips in the current visible set of toolbars.

Beginning with Microsoft Visio 2002, this setting corresponds to the **Show ScreenTips on toolbars** setting on the **Options** tab in the **Customize** dialog box (on the **Tools** menu, click **Customize**), and is shared between Visio 2002 and all Microsoft Office XP applications.

DistanceFrom property

Example

Returns the distance from one shape to another. Both shapes must be on the same page or in the same master.

Version added

2000

Syntax

doubleRet = *object*.**DistanceFrom**(*otherShape*, *flags*)

doubleRet	**Double**. A distance in internal drawing units with respect to the coordinate space defined by the parent shape.
object	Required. An expression that returns a **Shape** object.
otherShape	Required. The other **Shape** object involved in the comparison.
flags	Required **Integer**. Flags that influence the type of entries returned in results.

Remarks

The **DistanceFrom** property returns:

Zero and raises an exception if the shapes being compared are in different masters or on different pages.

Zero if the shapes being compared are overlapping.

Zero if one shape contains the other shape, or one shape is contained within the other shape.

The *flags* argument can be any combination of the values of the constants defined in the following table. These constants are also defined in **VisSpatialRelationFlags** in the Visio type library.

Constant	Value	Description
visSpatialIncludeHidden	&H10	Consider hidden Geometry sections. By default, hidden Geometry sections do not influence the result.
visSpatialIgnoreVisible	&H20	Do not consider visible Geometry sections. By default, visible Geometry sections influence the result.

Use the NoShow cell to determine whether a Geometry section is hidden or visible. Hidden Geometry sections have a value of TRUE and visible Geometry sections have a value of FALSE in the NoShow cell.

If *object* or *otherShape* has no geometry, or if *flags* excludes consideration of all geometry of either shape, then the **DistanceFrom** property returns a large number (1E+30) which should be construed as infinite.

The **DistanceFrom** property does not consider the width of a shape's line, shadows, line ends, control points, or connection points when comparing two shapes.

DistanceFromPoint property

Example

Returns the distance from a shape to a point.

Version added

2000

Syntax

doubleRet = *object*.**DistanceFromPoint**(*x*, *y*, *flags*, [*pPat*

doubleRet	**Double**. A distance in internal drawing units with respec point (*x*,*y*).
object	Required. An expression that returns a **Shape** object.
x	Required **Double**. An *x*-coordinate.
y	Required **Double**. A *y*-coordinate.
flags	Required **Integer**. Flags that influence the type of entrie
pPathIndex	Optional **Variant**. Identifies the point on the shape in co
pCurveIndex	Optional **Variant**. Identifies the point on the shape in co
pt	Optional **Variant**. Identifies the point on the shape in co *pCurveIndex*.

Remarks

The (*x*,*y*) point is expressed in internal drawing units (inches in the drawing) with respect to the coordinate space defined by the sheet immediately containing ThisShape.

The *pPathIndex*, *pCurveIndex*, and *pt* arguments optionally return values that identify the point the returned distance is measured from. Call that point (*xOnThis*,*yOnThis*). It lies along the *c*'th curve of ThisShape's *p*'th path and can be determined by:

ThisShape.Paths(*pPathIndex).Item(*pCurveIndex).Point

You can use the **PointAndDerivatives** method instead of the **Point** method if you want to find the first and second derivatives at position *t* along the curve.

If *pPathIndex* or *pCurveIndex* is not **Null**, an **Integer** (type VT_I4) is returned. If *p* isn't **Null**, it returns a **Double** (type VT_R8).

The **DistanceFromPoint** property considers guides to have extent and considers a shape's filled areas and paths.

The *flags* argument can be any combination of the values of the constants defined in the following table. These constants are also defined in **VisSpatialRelationFlags** in the Visio type library.

Constant	Value	Description
visSpatialIncludeHidden	&H10	Consider hidden Geometry sections. By default, hidden Geometry sections do not influence the result.
visSpatialIgnoreVisible	&H20	Do not consider visible Geometry sections. By default, visible Geometry sections influence the result.

Use the NoShow cell to determine whether a Geometry section is hidden or visible. Hidden Geometry sections have a value of TRUE and visible Geometry sections have a value of FALSE in the NoShow cell.

If *object* has no geometry, or if *flags* excludes consideration of all geometry, then the **DistanceFromPoint** property returns a large number (1E+30) which should be interpreted as infinite.

The **DistanceFromPoint** property does not consider the width of a shape's line, shadows, line ends, control points, or connection points when computing its result.

Document property

See also

Gets the **Document** object that is associated with an object.

Version added

2.0

Syntax

objRet = *object*.**Document**

objRet	The **Document** object that contains the object.
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

The **Document** property of a docked stencil window returns a **Document** object for the stencil that is currently at the top of the window. If another stencil replaces the first in the top position, the first stencil's document is closed so the reference to it becomes invalid. For best results, assume that document references to docked stencils are not persistent.

If a **Window** object shows no documents are open, then no document is returned and no exception is raised. Your solution should check for **Nothing** returned after retrieving the **Document** property of a **Window** object.

Documents property

object;DAR_Objects_(A-M)_1015.htm">

Returns the **Documents** collection for a Microsoft Visio instance.

Version added

2.0

Syntax

objsRet = *object*.**Documents**

objsRet	The **Documents** collection of the **Application** object.
object	Required. An expression that returns the **Application** object that owns the collection.

Remarks

You can iterate through a **Documents** collection by using the **Count** property to retrieve the number of documents in the collection. You can use the **Item** property to retrieve individual elements from a collection.

DocumentSheet property

See also Example

Returns a **Shape** object whose cells represent properties of the document.

Version added

2000

Syntax

objRet = *object*.**DocumentSheet**

objRet	A **Shape** object.
object	Required. An expression that returns a **Document** object.

DrawingPaths property

Example

Gets or sets the paths where Microsoft Visio looks for drawings.

Version added

4.0

Syntax

strRet = *object*.**DrawingPaths**

object.**DrawingPaths** = *pathsStr*

object	Required. An expression that returns an **Application** object.
strRet, pathsStr	**String**. A text string containing a list of folders.

Remarks

To indicate more than one folder, separate individual items in the path string with semicolons.

The string passed to and received from the **DrawingPaths** property is the same

string shown on the **File Paths** tab in the **Options** dialog box (click **Options** on the **Tools** menu). This string is stored in **HKEY_CURRENT_USER\Software\Microsoft\Visio\application\DrawingsP**

Visio looks for drawings in all paths named in the **DrawingPaths** property and all the subfolders of those paths. If you pass the **DrawingPaths** property to the **EnumDirectories** method, it returns a complete list of fully qualified paths in which Visio looks.

If a path is not fully qualified, Visio looks for the folder in the folder that contains the Visio program files (*appObj*.**Path**). For example, if the Visio executable file is installed in c:\Visio, and the **DrawingPaths** property is "Drawings;d:\Drawings", Visio looks for add-ons in both c:\Visio\Drawings and d:\Drawings.

DropActions property

See also Example

Defines special actions to be performed on shapes created using a master shortcut.

Version added

2000

Syntax

strRet = *object*.**DropActions**

object.**DropActions** = *strExpression*

strRet	**String**. One or more actions separated by semicolons. See example in Remarks.
object	Required. An expression that returns a **MasterShortcut** object.
strExpression	Required **String**. One or more actions separated by semicolons. See example in Remarks.

Remarks

When you drop a master shortcut onto a drawing page, Microsoft Visio applies any drop actions in the shortcut to the newly created shape. Each drop action defines a particular value or formula to be assigned to a particular cell in the new shape.

Because drop actions are defined by the shortcut, not the target master, it is possible to create several shortcuts that refer to the same target master, but which produce very different effects when dropped onto the drawing page.

The **DropActions** property can be blank, or can define a series of one or more individual drop actions. Actions are separated by semicolons (;). Each action consists of the name of the cell to change, followed by the formula to apply to that cell, separated by an equals sign (=). For example:

```
User.SubType=3; FillForegnd=7; Sheet2!Width=(ThePage
```

The application does not validate drop actions until they are applied to a new shape. If the **DropActions** property contains syntax errors or invalid cell names, the offending actions are ignored. However, if the application is running in developer mode, an error message is displayed, identifying the invalid action and the cause of the error. When using shortcut drop actions in your code, always test your shortcuts in developer mode to make sure the drop actions do not contain errors. To run in developer mode, on the **Tools** menu, click **Options**, click the **Advanced** tab, and then select the **Run in developer mode** check box.

DynamicGridEnabled property

Example

Determines whether the dynamic grid is enabled or not.

Version added

2002

Syntax

boolRet = *object*.**DynamicGridEnabled**

object.**DynamicGridEnabled** = *boolExpression*

boolRet	**Boolean**. **True** if dynamic grid is enabled; **False** if it is not enabled.
object	Required. An expression that returns a **Document** object.
boolExpression	Required **Boolean**. **True** to enable dynamic grid; **False** to disable it.

EditCopy property

See also Example

Returns a master that is open for editing and originally copied from this master.

Version added

2002

Syntax

objRet = *object*.**EditCopy**

objRet	The **Master** object that is open for editing and that was originally copied from *object*.
object	Required. An expression that returns a **Master** object.

Remarks

If there is no master associated with *object* that is open for editing, the **EditCopy** property returns **Nothing**.

EmailRoutingData property

See also Example

Returns e-mail routing data for a document.

Version added

2002

Syntax

varRet = object.**EmailRoutingData**

varRet	**Variant**. An array containing the e-mail routing data for a document.
object	Required. An expression that returns a **Document** object.

Remarks

The data contained in *varRet* is the equivalent of the data contained in the **Routing Slip** dialog box (on the **File** menu, point to **Send To**, and then click **Routing Recipient**).

Enabled property

See also

Determines whether or not an object is currently enabled.

Version added

4.0

Syntax

intVal = *object*.**Enabled**

object.**Enabled** = *intExpression*

intVal	**Integer**. **False** (0) if the object is disabled; **True** (-1) if it is enabled.
object	Required. An expression that returns an object in the **Applies to** list.
intExpression	Required **Integer**. **False** (0) to disable the object; **True** (non-zero) to enable the object.

Remarks

You can get and set the **Enabled** property of an **Event** object. An **Event** object that is disabled doesn't perform its action when its event occurs.

An add-on implemented by an executable (EXE) file always reports itself as enabled. An add-on implemented by a Visio Solutions Library (VSL) file reports itself as enabled or disabled according to the enabling policy that the VSL file has registered for that add-on.

You can't tell an add-on to enable or disable itself. Visio will not send a run message to a disabled add-on. The name of a disabled add-on on a Visio menu appears dimmed or gray.

End property (Characters object)

Example

Returns or sets the ending index of the indicated **Characters** object representing a range of text in a shape.

Version added

3.0

Syntax

intRet = *object*.**End**

object.**End** = *intExpression*

intRet	**Integer**. The current ending index of the **Characters** object.
object	Required. An expression that returns a **Characters** object.
intExpression	Required **Integer**. The new ending index of the **Characters** object.

Remarks

The **End** property determines the end of the text range represented by a

Characters object. The value of the **End** property is an index that represents the boundary between two characters, similar to an insertion point in text. Like selected text in a drawing window, a **Characters** object represents the sequence of characters that are affected by subsequent actions, such as the **Cut** or **Copy** method. When you retrieve a **Characters** object, its current text range includes all the shape's text. You can change the text range by setting the **Characters** object's **Begin** and **End** properties. Changing the text range of a **Characters** object has no effect on the text of the corresponding shape.

The **End** property can have a value from zero (0) to the value of the **CharCount** property for the corresponding shape. An index of 0 is positioned before the first character in the shape's text. An index that is the same as the **CharCount** property is positioned after the last character in the shape's text. If you specify a value less than 0, Visio uses 0. If you specify a value that is inside the expanded characters of a field, Visio sets the value of the **End** property to the end of the field.

The value of the **End** property must always be greater than or equal to the value of the **Begin** property. If you attempt to set the value of the **End** property to a value lower than the **Begin** property, Visio sets both the **End** and **Begin** properties to the value specified for the **End** property.

End property (Curve object)

Returns the end point of a **Curve** object.

Version added

5.0

Syntax

retVal = *object*.**End**

retVal	**Double**. Ending value of a **Curve** object's parameter domain.
object	Required. An expression that returns a **Curve** object.

Remarks

The **End** property of a **Curve** object returns the end point of a curve. A **Curve** object describes itself in terms of its parameter domain, which is the range [Start(),End()] where End() produces the curve's end point.

Error property

See also Example

Gets the error code generated by the last evaluation of a cell's formula.

Version added

2.0

Syntax

intRet = *object*.**Error**

intRet	**Integer**. The error code from the last evaluation of the cell's formula.
object	Required. An expression that returns a **Cell** object.

Remarks

When you evaluate a cell's formula, an error code is generated along with the result. The **Error** property allows you to access this error code. Constants for valid error codes are declared by the Visio type library and begin with **visError**.

Event property

Example

Gets or sets the event code of an **Event** object—an event-action pair. When the event occurs, the action is performed.

Version added

4.0

Syntax

intRet = *object*.**Event**

object.**Event** = *eventCode*

intRet	**Integer**. The current event code.
object	Required. An expression that returns an **Event** object.
eventCode	Required **Integer**. The new event code.

Remarks

If the action code of the **Event** object is **visActCodeRunAddon**, the event also specifies the target of the action and the arguments to send to the target. This

information is stored in the **Target** and **TargetArgs** properties, respectively.

If the action code of the **Event** object is **visActCodeAdvise**, the event also specifies the object to receive event notifications (sometimes called the sink object) and arguments to send to the sink object along with the notification.

Event codes are declared by the Visio type library. They are prefixed with "**visEvt**" and are listed in event topics in this Automation Reference. For a list of event codes, see Event codes.

A program can use the **Trigger** method to cause an **Event** object's action to be performed without waiting for the event to occur.

EventInfo property

Example

Gets additional information associated with an event, if any exists.

Version added

4.0

Syntax

strRet = *object*.**EventInfo**(*eventSeqNum*)

strRet	**String**. Additional information about the event.
object	Required. An expression that returns an **Application** object.
eventSeqNum	Required **Long**; **visEvtIDMostRecent** (0) for information about the most recently fired event, or the sequence number of the event to examine.

Remarks

When Visio fires an event, there are a small number of events for which additional information is available. These events are **BeforeDocumentSaveAs**, **DocumentSavedAs**, **EnterScope**, **ExitScope**, **MarkerEvent**, **ShapesDeleted**,

and **ShapeChanged**. Use the application's **EventInfo** property to obtain this information, when available.

The **EventInfo** property returns the following:

A string whose contents are specific to the event in question, if the event does record extra information.

An empty string if an event does not record extra information.

An error if Visio no longer has information for the specified event.

For details about the contents of the **EventInfo** property for an event, see the specific event topic.

If an event target queries the **EventInfo** property immediately after being triggered, the most recent event and the event whose sequence number was passed to the target are the same. However, if the target is an add-on implemented by an executable (EXE) file, this may not be the case because the executable file and Visio are separate tasks that aren't modal with respect to each other.

Note Event handlers that use the Microsoft Visual Basic for Applications **WithEvents** variable only have access to the most recent event and must use **visEvtIDMostRecent**.

To ensure that the information returned by the **EventInfo** property is associated with the same event that triggered the add-on, the executable file can pass <sequence number> as an argument to the **EventInfo** property. You can obtain the sequence number of an event in the following ways:

If the **Action** property of the **Event** object returns **visActCodeRunAddon**, then the command line string passed to the add-on contains a substring of the form "/eventid=<sequence number>".

> **Note** Even though the substring is labeled with "/eventid," don't confuse the <sequence number> passed in the command line string with the **ID** property of the firing **Event** object, which identifies the **Event** object in its **EventList** collection. The number being passed is actually the firing sequence number.

If the **Action** property of the **Event** object returns **visActCodeAdvise**, the

sequence number is passed as an argument to the **VisEventProc** procedure implemented by the target object.

EventList property

Returns the **EventList** collection of an object or the **EventList** collection that contains an **Event** object.

Version added

4.0

Syntax

objRet = *object*.**EventList**

objRet	The **EventList** collection.
object	Required. An expression that returns an object from the **Applies to** list that owns the collection.

EventsEnabled property

Determines whether a Microsoft Visio instance fires events.

Version added

4.5

Syntax

intRet = *object*.**EventsEnabled**

object.**EventsEnabled** = *intExpression*

intRet	**Integer**. **False** (0) if event firing is disabled; **True** (-1) if event firing is enabled.
object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. **False** (0) to disable event firing; **True** (non-zero) to enable event firing.

Remarks

If the **EventsEnabled** property is **False**, Visio does not fire events, run add-ons,

or execute Microsoft Visual Basic for Applications code when evaluating RUNADDON operands in cell formulas.

By default, the **EventsEnabled** property is **True** when an instance of Visio starts.

You may want to disable event firing if you have code behind events such as **DocumentOpened** or **DocumentCreated** that does not work properly, or to prevent the incorporation of a virus into a document. Events will not fire until the **EventsEnabled** property is set to **True**.

To set the **EventsEnabled** property to **False** in another way:

On the **Tools** menu, click **Options**.

In the **Options** dialog box, click the **Advanced** tab.

Clear the **Enable Automation events** check box.

ExtraInfo property

Example

Returns or sets extra URL request information used to resolve the hyperlink's URL.

Version added

5.0

Syntax

strRet = *object*.**ExtraInfo**

object.**ExtraInfo** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Hyperlink** object.
stringExpression	Required **String**. The new value for the field.

Remarks

Setting the **ExtraInfo** property of a shape's **Hyperlink** object is optional, and is equivalent to setting the value of the ExtraInfo cell in the shape's Hyperlink.Row

row.

You might, for example, set the **Hyperlink** object's **ExtraInfo** property to the coordinates of an image map, the contents of a form, or a file name.

If the **ExtraInfo** property you provide contains reserved characters other than spaces, you must input the escape character "%" and the character's hex equivalent. For example:

For "NAME=John Smith", set the **ExtraInfo** property to "NAME=John Smith" because the extra information contains spaces, but no reserved characters.

For "PATH=C:\TEMP", set **ExtraInfo** property to "PATH=C%3A%5CTEMP" because of the reserved characters.

FaceID property

Gets or sets the icon for an item.

Version added

2000

Syntax

intLong = *object*.**FaceID**

object.**FaceID** = *intLong*

intLong	Required **Integer**. Zero or more when the button uses one of the Visio standard built-in images. Negative one (-1) when the button has no icon, or has a custom image.
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

You can use any of the constants prefixed with **visIconIX** that are declared by

the Visio type library in **VisUIIconIDs**.

The **FaceID** property determines a button's icon, but not its function. Use the **CmdNum** property of a **ToolbarItem** or **MenuItem** object to set a button's function.

The **FaceID** property is the same as the **TypeSpecific1** property when the **CtrlType** property is type **visCtrlTypeBUTTON**, which is declared in the Visio type library in **VisUICtrlTypes**.

FieldCategory property

Example

Returns the field category for a field represented by an object.

Version added

3.0

Syntax

intRet = object.**FieldCategory**

intRet	**Integer**. The field category.
object	Required. An expression that returns a **Characters** object.

Remarks

If the **Characters** object does not contain a field or contains non-field characters, the **FieldCategory** property returns an exception. Check the **IsField** property of the **Characters** object before getting its **FieldCategory** property.

Field categories correspond to those in the **Category** list in the **Field** dialog box (click **Field** on the **Insert** menu).

To add a custom field, use the **AddCustomField** method.

The following constants for field categories are declared by the Visio type library in **VisFieldCategories**.

Constant	Value
visFCatCustom	0
visFCatDateTime	1
visFCatDocument	2
visFCatGeometry	3
visFCatObject	4
visFCatPage	5

FieldCode property

Example

Returns the field code for a field represented by an object.

Version added

3.0

Syntax

intRet = object.**FieldCode**

intRet	**Integer**. The field code.
object	Required. An expression that returns a **Characters** object.

Remarks

If the **Characters** object does not contain a field or contains non-field characters, the **FieldCode** property returns an exception. Check the **IsField** property of the **Characters** object before getting its **FieldCode** property.

Field codes correspond to the fields in the **Field** list in the **Field** dialog box (click **Field** on the **Insert** menu).

The following constants for field codes are declared by the Visio type library in **VisFieldCodes**.

- Constants for field codes

FieldFormat property

Example

Returns the field format for a field represented by an object.

Version added

3.0

Syntax

intRet = object.**FieldFormat**

intRet	**Integer**. The field format.
object	Required. An expression that returns a **Characters** object.

Remarks

If the **Characters** object does not contain a field or contains non-field characters, the **FieldFormat** property returns an exception. Check the **IsField** property of the **Characters** object before getting its **FieldFormat** property.

Field formats correspond to the formats in the **Format** list in the **Field** dialog box (click **Field** on the **Insert** menu).

The following constants for field formats are declared by the Visio type library in **VisFieldFormats**.

- Constants for field formats

FieldFormula[U] property

Example

Returns the formula of the custom field represented by an object.

Version added

3.0

Syntax

strRet = object.**FieldFormula**

strRet	**String**. The formula of the custom field.
object	Required. An expression that returns a **Characters** object.

Remarks

If the **Characters** object does not contain a field or contains non-field characters, or if the field is not a custom field, the **FieldFormula** property returns an exception. Check the **IsField** and **FieldCategory** properties of the **Characters** object before getting its **FieldFormula** property.

The formula returned by the **FieldFormula** property corresponds to the formula

that appears in the **Custom formula** box in the **Field** dialog box (click **Field** on the **Insert** menu).

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **FieldFormula** property to get a formula using local syntax. Use the **FieldFormulaU** property to get a formula using universal syntax.

FillBasedOn property

Example

Gets or sets the fill style on which the **Style** object is based.

Version added

4.0

Syntax

strVal = *object*.**FillBasedOn**

object.**FillBasedOn** = *styleName*

strVal	**String**. The name of the current fill style.
object	Required. An expression that returns a **Style** object.
styleName	Required **String**. The name of the new fill style.

Remarks

To base a style on no style, set the **FillBasedOn** property to a zero-length string ("").

FillStyle property

Returns or sets the fill style for an object.

Version added

2.0

Syntax

strRet = *object*.**FillStyle**

object.**FillStyle** = *stringExpression*

strRet	**String**. The current fill style.
object	Required. An expression that returns a **Selection** or **Shape** object that has or gets the fill style.
stringExpression	Required **String**. The name of the fill style to apply.

Remarks

Setting the **FillStyle** property is equivalent to selecting a style from the **Fill Style** list on the **Format Shape** toolbar.

Setting a style to a nonexistent style generates an error. Setting one type of style to another type (for example, setting the **FillStyle** property to a line style) does nothing. Setting one type of style to another type that has more than one set of attributes changes only the appropriate attributes. For example, setting the **FillStyle** property to a style with line, text, and fill attributes changes only the fill attributes.

To preserve a shape's local formatting, use the **FillStyleKeepFmt** property.

Beginning with Microsoft Visio 2002, a zero-length string ("") will cause the master's style to be reapplied to the selection or shape. (Earlier versions generate a "no such style" exception.) If the selection or shape has no master, its style remains unchanged. Setting *stringExpression* to a zero-length string ("") is equivalent to selecting ***Use master's format*** in the **Text style**, **Line style**, or **Fill style** list in the **Style** dialog box (on the **Format** menu, click **Style**).

FillStyleKeepFmt property

Example

Applies a fill style to an object while preserving local formatting.

Version added

2.0

Syntax

object.**FillStyleKeepFmt** = *stringExpression*

object	Required. An expression that returns a **Selection** or **Shape** object to which the fill style is applied.
stringExpression	Required **String**. The name of the fill style to apply.

Remarks

Setting the **FillStyleKeepFmt** property is equivalent to selecting the **Preserve local formatting** check box in the **Style** dialog box (click **Style** on the **Format** menu).

Setting a style to a nonexistent style generates an error. Setting one type of style

to another type (for example, setting the **FillStyleKeepFmt** property to a line style) does nothing. Setting one type of style to another type that has more than one set of attributes changes only the appropriate attributes (for example, setting the **FillStyleKeepFmt** property to a style with line, text, and fill attributes changes only the fill attributes).

Beginning with Microsoft Visio 2002, an empty string ("") will cause the master's style to be reapplied to the selection or shape. (Earlier versions generate a "no such style" exception.) If the selection or shape has no master, its style remains unchanged. Setting *stringExpression* to an empty string is equivalent to selecting ***Use master's format*** in the **Text style**, **Line style**, or **Fill style** lists in the **Style** dialog box (on the **Format** menu, click **Style**).

FilterPaths property

Gets or sets the path where Microsoft Visio looks for import and export filters.

Version added

4.0

Syntax

strRet = *object*.**FilterPaths**

object.**FilterPaths** = *pathsStr*

strRet	**String**. A text string containing the name of a folder.
object	Required. An expression that returns an **Application** object.
pathsStr	Required **String**. A text string containing the new folder name.

Remarks

The string passed to and received from the **FilterPaths** property is the same string shown on the **File Paths** tab in the **Options** dialog box (click **Options** on the **Tools** menu). This string is stored in

HKEY_CURRENT_USER\Software\Microsoft\Visio\application\FiltersPath

Unlike similar properties such as **AddonPaths** and **TemplatePaths**, you can name only one path in the **FilterPaths** property and Visio will not look for filters in the subfolders of the path you specify.

If a path is not fully qualified, Visio looks for the folder in the folder that contains the Visio program files (*appObj*.**Path**).

Flags property

Example

Gets or sets the flags that specify how you use a **Color** object.

Version added

4.0

Syntax

intRet = *object*.**Flags**

object.**Flags** = *intVal*

intRet	**Integer**. The current value of the color's flags component.
object	Required. An expression that returns a **Color** object.
intVal	Required **Integer**. The new value of the color's flags component.

Remarks

The **Flags** property of a **Color** object corresponds to the **peFlags** member of a Microsoft Windows **PALETTEENTRY** data structure. For details, search for

"**PALETTEENTRY**" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

Flavor property

Microsoft Visio no longer supports this property and ignores it in code. The Visio user interface uses only the Microsoft Office toolbar set.

Fonts property

Example

Returns the **Fonts** collection of a **Document** object.

Version added

4.0

Syntax

objRet = *object*.**Fonts**

objRet	Required. An expression that returns a document's **Fonts** collection.
object	Required. An expression that returns a **Document** object.

FooterCenter property

Example

Contains the text string that appears in the center portion of a document's footer.

Version added

2002

Syntax

strRet = *object*.**FooterCenter**

object.**FooterCenter** = *stringExpression*

strRet	**String**. The text in the center portion of the footer.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. New text for the center portion of the footer.

Remarks

You can also set this value in the **Center** box under **Footer** in the **Header and Footer** dialog box (on the **View** menu, click **Header and Footer**).

Both *strRet* and *strExpression* can contain escape codes that represent data. These escape codes can be concatenated with other text. For a list of valid escape codes you can use with the **FooterCenter** property, see the FooterLeft property.

Example

The following macro is used to place a string containing the current page number and total number of pages into the center portion of a document's footer.

```
Sub SetFooterCenter()
    Dim szFooter as String
    'Build footer string
    szFooter = "Page &p of &P"
    'Set footer of current document
    ThisDocument.FooterCenter = szFooter
End Sub
```

If this is a one-page document, the center portion of the footer contains "Page 1 of 1" after running this macro.

FooterLeft property

Example

Contains the text string that appears in the left portion of a document's footer.

Version added

2002

Syntax

strRet = *object*.**FooterLeft**

object.**FooterLeft** = *stringExpression*

strRet	**String**. The text in the left portion of the footer.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. New text for the left portion of the footer.

Remarks

You can also set this value in the **Left** box under **Footer** in the **Header and Footer** dialog box (on the **View** menu, click **Header and Footer**).

Both *strRet* and *strExpression* can contain escape codes that represent data. These escape codes can be concatenated with other text.

Following is a list of valid escape codes for document footers and headers.

Escape code	Description
&p	Page number
&t or &T	Current time
&d (short version) or &D (long version)	Current date
&&	Ampersand
&e	File name extension
&f	File name
&f&e	File name and extension
&n	Page name
&P	Total printed pages

Example

The following macro is used to place a string containing the current date into the left portion of a document's footer.

```
Sub SetFooterLeft()
    Dim szFooter as String
    'Build footer string
    szFooter = "The Date is " & "&D"
    'Set footer of current document
    ThisDocument.FooterLeft = szFooter
End Sub
```

If the date was May 4, 2002, the left portion of the footer contains "The Date is Thursday, May 4, 2002" after running this macro.

FooterMargin property

Example

Gets or sets the margin of a document's footer.

Version added

2002

Syntax

retVal = object.**FooterMargin** ([*unitsNameOrCode*])

object.**FooterMargin** ([*unitsNameOrCode*]) = *newVal*

retVal	**Double**. The margin size.
object	Required. An expression that returns a **Document** object.
unitsNameOrCode	Optional **Variant**. The units to use when retrieving or setting the cell's value. Defaults to internal drawing units (inches).
newVal	Required **Double**. The new margin size.

Remarks

If *unitsNameorCode* is not provided, the **FooterMargin** property will default to

internal drawing units.

You can also set this value in the **Margin** box under **Footer** in the **Header and Footer** dialog box (on the **View** menu, click **Header and Footer**).

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

FooterRight property

Example

Contains the text string that appears in the right portion of a document's footer.

Version added

2002

Syntax

strRet = *object*.**FooterRight**

object.**FooterRight** = *stringExpression*

strRet	**String**. The text in the right portion of the footer.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. New text for the right portion of the footer.

Remarks

You can also set this value in the **Right** box under **Footer** in the **Header and Footer** dialog box (on the **View** menu, click **Header and Footer**).

Both *strRet* and *strExpression* can contain escape codes that represent data. These escape codes can be concatenated with other text. For a list of valid escape codes you can use with the **FooterRight** property, see the FooterLeft property.

ForeignData property

Example

Returns metafile, bitmap, or OLE data for a shape that represents a foreign object.

Version added

2002

Syntax

retVal = object.**ForeignType**

retVal	**Byte**. An array containing metafile, bitmap, or OLE data for the shape.
object	Required. An expression that returns a **Shape** object.

Remarks

To determine whether a shape represents a foreign object, use the **ForeignType** property.

ForeignType property

Example

Returns the subtype of a **Shape** object that represents a foreign object.

Version added

4.1

Syntax

retVal = object.**ForeignType**

Remarks

If the **Type** property of a **Shape** object returns any value other than **visTypeForeignObject**, the **ForeignType** property returns the same value as the **Shape** object's **Type** property. If the **Type** property of a **Shape** object returns **visTypeForeignObject**, the **ForeignType** property returns a combination of the following values.

Constant	Value
visTypeMetafile	&H0010

visTypeBitmap	&H0020
visTypeIsLinked	&H0100
visTypeIsEmbedded	&H0200
visTypeIsControl	&H0400
visTypeIsOLE2	&H8000

If the shape represents an OLE 2.0 embedded object, for example, its **ForeignType** property is &H8200.

Formula[U] property

Returns or sets the formula for a **Cell** object.

Version added

2.0

Syntax

strRet = *object*.**Formula**

object.**Formula** = *stringExpression*

strRet	**String**. The cell's formula.
object	Required. An expression that returns a **Cell** object.
stringExpression	Required **String**. The new formula for the cell.

Remarks

If a cell's formula is protected with the GUARD function, you must use the **FormulaForce** property to change the cell's formula.

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **Formula** property to get a cell's formula string in local syntax or to set it using a mix of local and universal syntax. Use the **FormulaU** property to get or parse the formula using universal syntax. When using **FormulaU**, the decimal point is always "." and the delimiter is always "," and universal unit strings must be used (for details on universal strings, see About units of measure).

FormulaForce[U] property

Example

Sets the formula in a **Cell** object, even if the formula is protected with a GUARD function.

Version added

2.0

Syntax

object.**FormulaForce** = *stringExpression*

object	Required. An expression that returns a **Cell** object.
stringExpression	Required **String**. The new formula for the cell.

Remarks

Many of the SmartShapes symbols provided with Visio have guarded cells to maintain their smart behavior. When you change the formula in a guarded cell, the shape's behavior might change in unexpected ways.

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles,

pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **FormulaForce** property to get or parse the formula using local syntax. Use the **FormulaForceU** property to get or parse the formula using universal syntax.

Frame property

Example

Returns or sets the name of an HTML frame in the shape's **Hyperlink** object.

Version added

5.0

Syntax

strRet = *object*.**Frame**

object.**Frame** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Hyperlink** object.
stringExpression	Required **String**. The new value for the field.

Remarks

Setting the **Frame** property of a shape's **Hyperlink** object is optional and only applies when a Visio instance is open in a browser, for example, Microsoft Internet Explorer 3.0 or later.

Setting the **Frame** property is equivalent to setting the result of the Frame cell in the shape's Hyperlink.Row row.

FromCell property

Returns the cell from which a connection originates.

Version added

2.0

Syntax

objRet = *object*.**FromCell**

objRet	The **Cell** object from which the connection originates.
object	Required. An expression that returns a **Connect** object.

Remarks

A connection is defined by a reference in a cell in the shape from which the connection originates to a cell in the shape to which the connection is made. The **FromCell** property returns the **Cell** object for the cell from which the connection originates.

Following is a list of possible connections and the values of their related

FromCell properties.

- From the begin or end cell of a 1-D shape to…
- From the edge (a cell in the Alignment section) of a 2-D shape to…
- From an outward or inward/outward connection point cell of a 1-D shape to…
- From an outward or inward/outward connection point cell of a 2-D shape to…
- From a control point cell to…

FromConnects property

Example

Returns a **Connects** collection of the shapes connected to a shape.

Version added

4.5

Syntax

objRet = *object*.**FromConnects**

objRet	The **Connects** collection of shapes connected to this shape.
object	Required. An expression that returns a **Shape** object.

Remarks

The **FromConnects** property of a shape returns a **Connects** collection that contains every **Connect** object for which the shape is the **ToSheet** property. This tells you all the shapes connected to a shape.

To obtain a **Connects** collection that contains every **Connect** object for which the shape is the **FromSheet** property, use the shape's **Connects** property. This

tells you all the shapes to which the shape is connected.

FromPart property

Returns the part of a shape from which a connection originates.

Version added

2.0

Syntax

retVal = object.**FromPart**

retVal	**Integer**. The part of the shape where the connection originates.
object	Required. An expression that returns a **Connect** object.

Remarks

The following constants declared by the Visio type library show return values for the **FromPart** property.

Constant	Value
visConnectFromError	-1
visFromNone	0

visLeftEdge	1
visCenterEdge	2
visRightEdge	3
visBottomEdge	4
visMiddleEdge	5
visTopEdge	6
visBeginX	7
visBeginY	8
visBegin	9
visEndX	10
visEndY	11
visEnd	12
visFromPin	13
visFromAngle	14
visControlPoint	100 + zero-based row index (for example, **visControlPoint** = 100 if the control point is in row 0; **visControlPoint** = 101 if the control point is in row 1)

FromSheet property

Returns the shape from which a connection or connections originate.

Version added

2.0

Syntax

objRet = *object*.**FromSheet**

objRet	The **Shape** object from which the connections originate.
object	Required. An expression that returns the **Connect** object or **Connects** collection to examine.

Remarks

The **FromSheet** property for a **Connect** object is straightforward. It always returns the shape from which the **Connect** object originates.

A **Connects** collection represents several connections. If every connection represented by the collection originates from the same shape, the **FromSheet**

property for the collection returns that shape. Otherwise, the **FromSheet** property returns **Nothing** and does not raise an exception.

FullBuild property

Example

Returns the full build number of the running instance.

Version added

2002

Syntax

retVal = object.**FullBuild**

retVal	**Long**. The build number.
object	Required. An expression that returns an **Application** object.

Remarks

The format of the build number is described in the following table.

Bits	Description
0 - 15	Internal build number
16 - 20	Internal revision number

21 - 25	Minor version number
26 - 30	Major version number (Visio 2002 = 10)
31	Reserved

The build number of the running instance is written to the **FullBuildNumberCreated** property when a new document is created, and to the **FullBuildNumberEdited** property when a document is edited.

FullBuildNumberCreated property

Example

Returns the full build number of the instance used to create the document.

Version added

2002

Syntax

retVal = object.**FullBuildNumberCreated**

retVal	**Long**. The build number when the document was created.
object	Required. An expression that returns a **Document** object.

Remarks

The format of the build number is described in the following table.

Bits	Description
0 - 15	Internal build number
16 - 20	Internal revision number

21 - 25	Minor version number
26 - 30	Major version number (Visio 2002 = 10)
31	Reserved

FullBuildNumberEdited property

Example

Returns the full build number of the instance last used to edit the document.

Version added

2002

Syntax

retVal = object.**FullBuildNumberEdited**

retVal	**Long**. The build number when the document was last edited.
object	Required. An expression that returns a **Document** object.

Remarks

The format of the build number is described in the following table.

Bits	Description
0 - 15	Internal build number
16 - 20	Internal revision number

21 - 25	Minor version number
26 - 30	Major version number (Visio 2002 = 10)
31	Reserved

FullName property

Example

Returns the name of a document, including the drive and path.

Version added

2.0

Syntax

strRet = object.**FullName**

strRet	**String**. The file name of the document.
object	Required. An expression that returns a **Document** object.

Remarks

Use the **FullName** property to obtain a document's drive, folder path, and file name as one string. The returned value can include UNC drive names (for example, \\bob\leo).

GeometryCount property

Returns the number of Geometry sections for a shape.

Version added

2.0

Syntax

intRet = *object*.**GeometryCount**

intRet	**Integer**. The number of Geometry sections for the shape.
object	Required. An expression that returns a **Shape** object.

GestureFormatSheet property

Example

Returns a reference to a document's Gesture Format sheet, which contains the line, fill, and text formatting that is applied to shapes drawn on the page.

Version added

2000

Syntax

objRet = *object*.**GestureFormatSheet**

objRet	A **Shape** object that contains line, fill, and text formatting to be applied to shapes drawn on the page.
object	Required. An expression that returns a **Document** object.

Remarks

By default, a new shape inherits all its formatting from the document's default styles. However, if the Gesture Format sheet contains local formatting, that formatting is applied to the new shape. Use the **FillStyle**, **LineStyle**, and **TextStyle** properties to apply local formatting to the Gesture Format **Shape**

object.

Gesture Format sheet formatting does not apply to instances of masters, connectors, pasted objects, or embedded objects.

A document's Gesture Format sheet is cleared automatically when a document is opened.

If a user makes changes to any shape using shape formatting commands on the menus and toolbars, but no shapes are currently selected, this formatting is stored in the gesture format sheet and applied to new shapes the user draws.

GlueEnabled property

See also Example

Determines whether glue is enabled in the document.

Version added

2002

Syntax

retVal = *object*.**GlueEnabled**

object.**GlueEnabled** = *newVal*

retVal	**Boolean**. **True** if glue is enabled; otherwise **False**.
object	Required. An expression that returns a **Document** object.
newVal	Required **Boolean**. **True** to enable glue behavior; **False** to disable glue behavior.

Remarks

The value of the **GlueEnabled** property corresponds to the setting of the **Glue** check box on the **General** tab in the **Snap & Glue** dialog box (on the **Tools** menu, click **Snap & Glue**).

GlueSettings property

See also Example

Determines the objects that shapes glue to when glue is enabled in the document.

Version added

2002

Syntax

retVal = *object*.**GlueSettings**

object.**GlueSettings** = *newVal*

retVal	**VisGlueSettings**. The objects in a document that shapes glue to.
object	Required. An expression that returns a **Document** object.
newVal	Required **VisGluesettings**. The objects in a document that shapes glue to.

Remarks

The value of the **GlueSettings** property is equivalent to selecting options under **Glue to** on the **General** tab in the **Snap & Glue** dialog box (on the **Tools** menu, click **Snap & Glue**).

The **GlueSettings** property can be any combination of the following **VisGlueSettings** constants, which are declared in the Visio type library.

Constant	Value	Description

visGlueToNone	&H0	Glue is enabled but no other glue settings are on.
visGlueToGuides	&H1	Glue to guides.
visGlueToHandles	&H2	Glue to shape handles.
visGlueToVertices	&H4	Glue to shape vertices.
visGlueToConnectionPoints	&H8	Glue to connection points.
visGlueToGeometry	&H20	Glue to shape geometry.
visGlueToDisabled	&H8000	Disable glue.

Green property

Example

Gets or sets the intensity of the green component of a **Color** object.

Version added

4.0

Syntax

```
intRet = object.Green
object.Green = intVal
```

intRet	**Integer**. The current value of the color's green component.
object	Required. An expression that returns a **Color** object.
intVal	Required **Integer**. The new value of the color's green component.

Remarks

The **Green** property can be a value from 0 to 255.

A color is represented by red, green, and blue components. It also has flags that indicate how the color is to be used. These correspond to members of the Microsoft Windows **PALETTEENTRY** data structure. For details, search for "PALETTEENTRY" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

HeaderCenter property

Example

Contains the text string that appears in the center portion of a document's header.

Version added

2002

Syntax

strRet = *object*.**HeaderCenter**

object.**HeaderCenter** = *stringExpression*

strRet	**String**. The text in the center portion of the header.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. New text for the center portion of the header.

Remarks

You can also set this value in the **Center** box under **Header** in the **Header and Footer** dialog box (on the **View** menu, click **Header and Footer**).

Both *strRet* and *strExpression* can contain escape codes that represent data. These escape codes can be concatenated with other text. For a list of valid escape codes you can use with the **HeaderCenter** property, see the FooterLeft property.

Example

The following macro is used to place the string containing "Document Title" into the center portion of the document's header.

```
Sub SetHeaderCenter()
    'Set header of current document
    ThisDocument.HeaderCenter = "Document Title"
End Sub
```

After running this macro, "Document Title" is displayed in the center of the document header.

HeaderFooterColor property

Example

Specifies the color of the header and footer text.

Version added

2002

Syntax

colorRet = *object*.**HeaderFooterColor**

object.**HeaderFooterColor** = *colorVal*

colorRet	OLE_COLOR. The color of the header and footer text.
object	Required. An expression that returns a **Document** object.
colorVal	Required OLE_COLOR. The new color for the header and footer text.

Remarks

Valid values for an OLE_COLOR property within Visio can be one of the following:

&H00*bbggrr,* where *bb* is the blue value between 0 and 0xFF (255), *gg* the green value, and *rr* the red value.

&H800000*x, where xx* is a valid **GetSysColor** index.

For details about the **GetSysColor** function, search for "**GetSysColor**" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

The OLE_COLOR data type is used for properties that return colors. When a property is declared as OLE_COLOR, the **Properties** window will display a color-picker dialog box that allows the user to select the color for the property visually, rather than having to remember the numeric equivalent.

You can also set this value in the **Color** box in the **Choose Font** dialog box (on the **View** menu, click **Header and Footer, and then click Choose Font**).

Example

The following macro is used to assign the color blue to text in the header and footer.

```
Sub SetHeaderFooterColor()
    'Set color of the header of this
    'document to blue
    ThisDocument.HeaderFooterColor = &H00FF0000
End Sub
```

After running this macro, the header text will display blue text.

HeaderFooterFont property

Example

Specifies the font used for the header and footer text.

Version added

2002

Syntax

fontRet = *object*.**HeaderFooterFont**

object.**HeaderFooterFont** = *fontVal*

fontRet	An **IFontDisp** object that represents the font of the header and footer text.
object	Required. An expression that returns a **Document** object.
fontVal	Required **IFontDisp**. An **IFontDisp** object representing the new font for the header and footer text.

Remarks

COM provides a standard implementation of a font object with the **IFontDisp**

interface on top of the underlying system font support. The **IFontDisp** interface exposes a font object's properties and is implemented in the stdole type library as a **StdFont** object that can be created in Microsoft Visual Basic. The stdole type library is automatically referenced from all Visual Basic for Applications (VBA) projects in Visio.

To get information about the **StdFont** object that supports the **IFontDisp** interface:

On the **Tools** menu, point to **Macros**, and then click **Visual Basic Editor**.

On the **View** menu, click **Object Browser**.

In the **Project/Library** list, click **stdole**.

Under **Classes**, examine the class named **StdFont**.

For details about the **IFontDisp** interface, see the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

This is the equivalent of setting values in the **Font** box in the **Choose Font** dialog box (on the **View** menu, click **Header and Footer, and then click Choose Font**).

Example

The following sample code illustrates getting a reference to the current **Font** object and changing two of its attributes, its name and boldness.

```
Public Sub SetHeaderFooterFontNonBoldArial()
    Dim oStdFont As StdFont
    Set oStdFont = ThisDocument.HeaderFooterFont
    oStdFont.Name = "Arial"
    oStdFont.Bold = False
    Set ThisDocument.HeaderFooterFont = oStdFont
End Sub
```

After running this macro, the header and footer text are displayed in no-bold, Arial.

HeaderLeft property

Example

Contains the text string that appears in the left portion of a document's header.

Version added

2002

Syntax

strRet = *object*.**HeaderLeft**

object.**HeaderLeft** = *stringExpression*

strRet	**String**. The text in the left portion of the header.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. New text for the left portion of the header.

Remarks

You can also set this value in the **Left** box under **Header** in the **Header and Footer** dialog box (on the **View** menu, click **Header and Footer**).

Both *strRet* and *strExpression* can contain escape codes that represent data. These escape codes can be concatenated with other text. For a list of valid escape codes you can use with the **HeaderLeft** property, see the FooterLeft property

Example

The following macro is used to place a string containing the current date into the left portion of a document's header.

```
Sub SetHeaderLeft()
    Dim szHeader as String
    'Build header string
    szHeader = "The Date is " & "&D"
    'Set header of current document
    ThisDocument.HeaderLeft = szHeader
End Sub
```

If the date was May 4, 2002, the left portion of the header contains "The Date is Thursday, May 4, 2002" after running this macro.

HeaderMargin property

Example

Gets or sets the margin of a document's header.

Version added

2002

Syntax

retVal = object.**HeaderMargin** ([*unitsNameOrCode*])

object.**HeaderMargin** ([*unitsNameOrCode*]) = *newVal*

retVal	**Double**. The margin size.
object	Required. An expression that returns a **Document** object.
unitsNameOrCode	Optional **Variant**. The units to use when retrieving or setting the cell's value. Defaults to internal drawing units (inches).
newVal	Required **Double**. The new margin size.

Remarks

You can also set this value in the **Margin** box under **Header** in the **Header and**

Footer dialog box (on the **View** menu, click **Header and Footer**).

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

HeaderRight property

Example

Contains the text string that appears in the right portion of a document's header.

Version added

2002

Syntax

strRet = *object*.**HeaderRight**

object.**HeaderRight** = *stringExpression*

strRet	**String**. The text in the right portion of the header.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. New text for the right portion of the header.

Remarks

You can also set this value in the **Right** box under **Header** in the **Header and Footer** dialog box (on the **View** menu, click **Header and Footer**).

Both *strRet* and *strExpression* can contain escape codes that represent data. These escape codes can be concatenated with other text. For a list of valid escape codes you can use with the **HeaderRight** property, see the FooterLeft property.

Height property

Example

Gets the height of a menu set or toolbar.

Version added

2000

Syntax

intRet = *object*.**Height**

intRet	**Integer**. The height.
object	Required. An expression that returns a **MenuSet** or **Toolbar** object for which you want to get the height.

Remarks

If the object is docked or protected from resizing, the application returns an error.

Help property

See also Example

Gets or sets the help string for a shape.

Version added

3.0

Syntax

strRet = *object*.**Help**

object.**Help** = *strExpression*

strRet	**String**. The current help string.
object	Required. An expression that returns a **Shape** object.
strExpression	Required **String**. The new help string.

Remarks

Using the **Help** property is equivalent to entering a value in the **Help** box for a shape in the **Special** dialog box (click **Special** on the **Format** menu). The limit for a help string is 127 characters.

HelpContextID property

Example

Gets or sets the help context ID to be used by a menu or toolbar item.

Version added

4.0

Syntax

object.**HelpContextID** = *intVal*

intVal = *object*.**HelpContextID**

object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** object that has or gets the context ID.
intVal	**Integer**. The context ID of a topic in a help file.

Remarks

For Visio commands, the **HelpContextID** property is usually the same value as the **CmdNum** property, which contains the command ID. Command IDs are declared by the Visio type library and have the prefix **visCmd**.

By default, the value of the **HelpContextID** property is zero (0), which displays the Contents topic of the help file indicated by the **HelpFile** property.

If the value of the **HelpContextID** property is zero and the object's **CmdNum** property is set to one of the Visio command IDs, it uses the default help context ID from the built-in Visio user interface.

HelpFile property

Example

Gets or sets the help file to be used by a menu or toolbar item.

Version added

4.0

Syntax

object.**HelpFile** = *fileStr*

fileStr = *object*.**HelpFile**

object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** object that has or gets the help file.
fileStr	Required **String**. The name of the help file.

Remarks

If you provide a fully qualified path along with the name of the help file, the application searches the folders specified in the **HelpPaths** property of the **Application** object.

If **HelpFile** is null and the object's **CmdNum** property is set to one of the Visio command IDs, your program uses the default help file from the built-in Visio user interface.

Note Set the **HelpContextID** property of the object to display a particular topic within a help file.

HelpPaths property

Gets or sets the paths where Microsoft Visio looks for help files.

Version added

4.0

Syntax

strRet = *object*.**HelpPaths**

object.**HelpPaths** = *pathsStr*

strRet	**String**. A text string containing a list of folders where Visio looks for help files. Individual items are separated by semicolons.
object	Required. An expression that returns an **Application** object.
pathsStr	Required **String**. A text string containing a list of folders; to indicate more than one folder, separate individual items in the path string with semicolons.

Remarks

The string passed to and received from the **HelpPaths** property is the same string shown on the **File Paths** tab in the **Options** dialog box (click **Options** on the **Tools** menu). This string is stored in **HKEY_CURRENT_USER\Software\Microsoft\Visio\application\HelpPath**.

When the application looks for help files, it looks in all paths named in the **HelpPaths** property and all the subfolders of those paths. If you pass the **HelpPaths** property to the **EnumDirectories** method, it returns a complete list of fully qualified paths in which Visio looks.

If a path is not fully qualified, the application looks for the folder in the folder that contains the Visio program files (*appObj*.**Path**). For example, if the Visio executable file is installed in c:\Visio, and the **HelpPaths** property is "Help;d:\Help", the Visio application looks for help files in both c:\Visio\Help and d:\Help.

Hidden property

See also Example

Hides a master on a stencil or a style in the user interface.

Version added

2000

Syntax

intRet = *object*.**Hidden**

object.**Hidden** = *intExpression*

intRet	**Integer**. **True** (-1) if the object is hidden; otherwise **False** (0).
object	Required. An expression that returns a **Master** or **Style** object that is hidden.
intExpression	Required **Integer**. **True** (-1) to hide the object; otherwise **False** (0).

Remarks

A master that is hidden still appears in the **Drawing Explorer**.

HitTest property

Example

Determines if a given *x*,*y* position hits outside, inside, or on the boundary of a shape.

Version added

4.5

Syntax

intRet = *object*.**HitTest**(*x*, *y*, *tolerance*)

intRet	**Integer**. Any combination of the values of the constants prefixed by **visHit**. See Remarks.
object	Required. An expression that returns a **Shape** object.
x	Required **Double**. The *x*-coordinate to be tested for a hit.
y	Required **Double**. The *y*-coordinate to be tested for a hit.
tolerance	Required **Double**. How close *x*,*y* must be to a shape for a hit to occur.

Remarks

The **HitTest** property considers only visible geometry, and ignores hidden geometry.

Use internal drawing units (inches in the drawing) for the *x*, *y*, and *tolerance* values. These values should also be in, and with respect to, the coordinate space of the page, master, or group shape that contains the shape being hit tested.

The following are possible values of *intRet*, and are declared by the Visio type library in **VisHitTestResults**.

Constant	Value
visHitOutside	0
visHitOnBoundary	1
visHitInside	2

hwnd property

Example

Gets or sets the **HWND** field of the **MSG** structure being wrapped.

Version added

2002

Syntax

retVal = object.**hwnd**

object.**hwnd =** *newVal*

retVal	**Long**. A handle to the window that fired the **OnKeystrokeMessageForAddon** event.
object	Required. An expression that returns a **MSGWrap** object.
newVal	Required **Long**. The new window handle.

Remarks

The properties of the **MSGWrap** object correspond to the fields in the **MSG**

structure defined as part of the Microsoft Windows operating system.

For details, search for "MSG structure" on the Microsoft Developer Network (MSDN) Web site.

Hyperlink property

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property returned a **Hyperlink** object that represented a shape's hyperlink.

HyperlinkBase property

Example

Gets or sets the value of the **Hyperlink base** field in a **Document** object's properties.

Version added

5.0

Syntax

strRet = *object*.**HyperlinkBase**

object.**HyperlinkBase** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The new value of the field.

Remarks

Setting the **HyperlinkBase** property is equivalent to entering information in the **Hyperlink base** box in the **Properties** dialog box (click **Properties** on the **File**

menu).

Hyperlinks property

See also Example

Returns the **Hyperlinks** collection for a **Shape** object.

Version added

2000

Syntax

objsRet = object.**Hyperlinks**

objsRet	The **Hyperlinks** collection for a **Shape** object.
object	Required. An expression that returns a **Shape** object.

Icon property

Example

Returns the icon contained in a master, master shortcut, or window.

Version Added

2002

Syntax

```
objRet = object.Icon

object.Icon = objVal
```

objRet	An **IPictureDisp** object that represents an icon.
object	Required. An expression that returns a **Master**, **MasterShortcut**, or **Window** object.
objVal	An **IPictureDisp** object that represents a new icon.

Remarks

The **Icon** property returns and accepts only HICON files. Visio raises an

exception if *objExpression* contains a non-HICON file.

COM provides a standard implementation of a picture object with the **IPictureDisp** interface on top of the underlying system picture support. The **IPictureDisp** interface exposes a picture object's properties and is implemented in the stdole type library as a **StdPicture** object creatable within Microsoft Visual Basic. The stdole type library is automatically referenced from all Visual Basic for Application projects in Visio.

To get information about the **StdPicture** object that supports the **IPictureDisp** interface:

On the **Tools** menu, point to **Macros**, and then click **Visual Basic Editor**.

On the **View** menu, click **Object Browser**.

In the **Project/Library** list, click **stdole**.

Under **Classes**, examine the class named **StdPicture**.

For details about the **IPictureDisp** interface, see the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

Currently, only in-proc solutions can use the **Icon** property because the **IPictureDisp** interface cannot be marshaled.

IconSize property

Example

Gets or sets the size of a master icon.

Version added

2.0

Syntax

intRet = *object*.**IconSize**

object.**IconSize** = *newSize*

intRet	The current size of the master icon.
object	Required. An expression that returns a **Master** or **MasterShortcut** object.
newSize	The new size for the master icon.

Remarks

The following constants declared by the Visio type library show the possible

values for the **IconSize** property.

Constant	Value
visNormal	1
visTall	2
visWide	3
visDouble	4

IconUpdate property

Example

Determines whether a master icon is updated manually or automatically.

Version added

2.0

Syntax

intRet = *object*.**IconUpdate**

object.**IconUpdate** = *updateMode*

intRet	The current update mode for the master icon.
object	Required. An expression that returns a **Master** object.
updateMode	The new update mode for the master icon.

Remarks

The following constants declared by the Visio type library show the possible values for the **IconUpdate** property.

Constant	Value
visManual	0
visAutomatic	1

ID property

Example

Gets the ID of an object.

Version added

4.0

Syntax

intVal = *object*.**ID**

intVal	**Long**. The ID of the object.
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

The ID of a shape is unique only within the scope of the page or master. The ID of a page, master, or style is unique within the scope of the document.

If a shape, page, master, or style is deleted, future objects in the same scope may be assigned the same ID. Therefore persisting shape or style IDs in separate data

stores is generally not as sound as persisting unique IDs using the **UniqueID** property.

For **Shape** objects, you can use the **ID** property with methods such as **GetResults** and **PutResults** to get or set many cell values at once, possibly cells in many different shapes. To do this, you need to pass shape IDs to the methods. If you create shapes using the **DropMany** method, the method returns the IDs of the shapes it creates to your program.

For **Font** objects, the **ID** property corresponds to the number stored in the Font cell of the row in a shape's Character Properties section. For example, to apply the font named "Arial" to a shape's text, create a **Font** object representing "Arial" and get the ID of that font, then set the **CharProps** property of the **Shape** object to that ID.

The ID associated with a particular font varies from system to system or as fonts are installed and removed on a given system.

For **Window** objects, the **ID** property can be used with the **ItemFromID** property of a **Windows** collection to retrieve a **Window** object from a **Windows** collection without iterating through the collection. A **Window** object with a **Type** property of **visAnchorBarBuiltIn** returns an ID of **visWinIDCustProp**, **visWinIDDrawingExplorer**, **visWinIDPanZoom**, or **visWinIDSizePos**. A **Window** object with a **Type** property of **visAnchorBarAddon** returns an ID that is unique within its **Windows** collection for the lifetime of that collection. If a **Window** object has an ID of **visInvalWinID**, you cannot retrieve the **Window** object from its collection using the **ItemFromID** property.

For **Event** objects, the **ID** property uniquely identifies an **Event** object in its **EventList** collection. As long as a reference is held on an **EventList** collection, or on the source object of an **EventList** collection, you can cache the **ID** property of any **Event** object in the list. Even if other events are added to or removed from the list, the cached ID can be used later to identify the original event. If an event is persistent, its ID can be cached indefinitely. While the event with that ID might be removed, no new **Event** object in the same **EventList** collection is given the same ID.

IncludesFill property

Example

Indicates whether the style includes fill attributes.

Version added

4.0

Syntax

intRet = *object*.**IncludesFill**

object.**IncludesFill** = *intExpression*

intRet	**Integer**. **False** (0) if the object doesn't define fill attributes; **True** (-1) if it does.
object	Required. An expression that returns a **Style** object.
intExpression	Required **Integer**. **False** (0) to disable fill attributes; **True** (non-zero) to enable them.

Remarks

The **IncludesFill** property corresponds to the **Fill** check box under **Includes** in

the **Define Styles** dialog box (click **Define Styles** on the **Format** menu).

IncludesLine property

Example

Indicates whether the style includes line attributes.

Version added

4.0

Syntax

intRet = *object*.**IncludesLine**

object.**IncludesLine** = *intExpression*

intRet	**Integer**. **False** (0) if the object doesn't define line attributes; **True** (-1) if it does.
object	Required. An expression that returns a **Style** object.
intExpression	Required **Integer**. **False** (0) to disable line attributes; **True** (non-zero) to enable them.

Remarks

The **IncludesLine** property corresponds to the **Line** check box under **Includes**

in the **Define Styles** dialog box (click **Define Styles** on the **Format** menu).

IncludesText property

Example

Indicates whether the style includes text attributes.

Version added

4.0

Syntax

intRet = *object*.**IncludesText**

object.**IncludesText** = *intExpression*

intRet	**Integer**. **False** (0) if the object doesn't define text attributes; **True** (-1) if it does.
object	Required. An expression that returns a **Style** object.
intExpression	Required **Integer**. **False** (0) to disable text attributes; **True** (non-zero) to enable them.

Remarks

The **IncludesText** property corresponds to the **Text** check box under **Includes** in

the **Define Styles** dialog box (click **Define Styles** on the **Format** menu).

Index property

Example

Gets the ordinal position of an object in a collection.

Version added

2.0

Syntax

intRet = object.**Index**

intRet	**Integer** or **Long**. The index of the object within its collection. See Remarks for return data type information.
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

Object	Return data type
Addon, **Document**, **Event**, **Font**, **Layer**, **Master**, **MasterShortcut**, **Row**, **Section**, **Window**	**Integer**

Color, **Connect**, **Menu**, **MenuItem**, **Long**
Shape, **Style**, **Toolbar**, **ToolbarItem**

Most collections are indexed starting with 1 rather than zero (0), so the index of the first element is 1, the index of the second element is 2, and so forth. The index of the last element in a collection is the same as the value of that collection's **Count** property. You can iterate through a collection by using these index values. Adding objects to or deleting objects from a collection can change the index values of other objects in the collection.

There are some exceptions. The **Color** collection is indexed starting with 0. This is consistent with the numbering displayed next to the colors that appear in the **Color Palette** dialog box (click **Color Palette** on the **Tools** menu).

These collections are also indexed starting with 0: **AccelItems**, **AccelTables**, **MenuSets**, **MenuItems**, **Menus**, **ToolbarItems**, **Toolbars**, and **ToolbarSets**.

Index property (Page object)

Example

Gets or sets the ordinal position of a page in a **Pages** collection.

Version added

2002

Syntax

intRet = *object*.**Index**

object.**Index** = *intExpression*

intRet	**Integer**. The index of the page within its collection.
object	Required. An expression that returns a **Page** object.
intExpression	Required **Integer**. The new index of a page within its collection.

Remarks

The **Pages** collection is indexed starting with 1 rather than zero (0), so the index of the first element is 1, the index of the second element is 2, and so on. The

index of the last element in a collection is the same as the value of that collection's **Count** property. You can iterate through a collection by using these index values. Adding objects to or deleting objects from a collection can change the index values of other objects in the collection.

You may only assign a new index to a foreground page. Background pages are unordered. Use the **Background** property to determine if a given page is a background page.

Use the **BackPage** property to assign a background page to a foreground page or to another background page.

IndexInStencil property

See also Example

Contains the index of a master or master shortcut object within its stencil.

Version added

2000

Syntax

intRet = *object*.**IndexInStencil**

object.**IndexInStencil** = *intExpression*

intRet	**Integer**. The index of the object within its stencil.
object	Required. An expression that returns a **Master** or **MasterShortcut** object.
intExpression	Required **Integer**. The new index of the object within its stencil.

Remarks

Beginning with Visio 2000, the document stencil window shows all **Master** and **MasterShortcut** objects in a Visio document. The Visio object model exposes the **Master** and **MasterShortcut** objects in a **Document** object as two distinct collections. The index returned by a **Master** object is its index with respect to other **Master** objects in its **Document** object and is unrelated to the presence or absence of **MasterShortcut** objects in the document. The index returned by a **MasterShortcut** object is its index with respect to other **MasterShortcut** objects in its **Document** object and is unrelated to the presence or absence of

Master objects in the document.

Use the **IndexInStencil** property to maintain the relative order of **Master** and **MasterShortcut** objects when considered as a single collection.

InheritedFormulaSource property

Example

Returns the cell from which this cell inherited its formula.

Version added

2002

Syntax

objRet = *object*.**InheritedFormulaSource**

objRet	The **Cell** object that contains the formula that *object* inherited.
object	The **Cell** object that contains the formula.

Remarks

If the formula in *thisCell* is a local formula, then the **InheritedFormulaSource** property returns itself.

InheritedValueSource property

Example

Returns the cell from which this cell inherited its value.

Version added

2002

Syntax

objRet = *object*.**InheritedValueSource**

objRet	The **Cell** object that contains the value which *object* inherited.
object	The **Cell** object that contains the value.

Remarks

If the value in *object* is a local value, then the **InheritedValueSource** property returns itself.

InhibitSelectChange property

See also Example

Determines whether shapes are selected in the drawing window.

Version added

2002

Syntax

boolRet = *object*.**InhibitSelectChange**

object.**InhibitSelectChange** = *boolExpression*

boolRet	**Boolean**. **True** if shapes are not selected; otherwise, **False**.
object	**Required**. An expression that returns an **Application** object.
boolExpression	Required **Boolean**. **True** to not select shapes; otherwise, **False**.

Remarks

Use the **InhibitSelectChange** property to control shape selection and increase performance when dropping a series of shapes in the drawing window. When the **InhibitSelectChange** property is **True**, Visio does not select any shapes after they are dropped. Your solution, however, can select shapes.

Additionally, Visio attempts to preserve currently selected shapes whenever possible, unless shapes are deselected by the solution.

If a program neglects to turn the **InhibitSelectChange** property off (**False**) after turning it on, the Visio instance will turn it back off when the user performs an

operation.

InPlace property

See also Example

Specifies whether a window is open in-place, or whether a document is being viewed through a window that is open in-place.

Version added

2002

Syntax

intRet = *object*.**InPlace**

intRet	**Integer**. **True** (-1) if a window or document in a window is open in-place; otherwise, **False** (0).
object	Required. An expression that returns a **Document** or **Window** object.

Remarks

When the value of the **InPlace** property is **True** for a **Window** object, it means that the window is an in-place active window. It contains a document that is being edited in an OLE container application.

When the value of the **InPlace** property is **True** for a **Document** object, it means that the document is open in an in-place editing window in an OLE container application.

InstanceHandle32 property

Example

Gets the instance handle of the **Application** object for a 32-bit version of Microsoft Visio.

Version added

4.0

Syntax

longRet = *object*.**InstanceHandle32**

longRet	**Long**. The instance handle of the object (a 4-byte value).
object	Required. An expression that returns an **Application** object.

Remarks

Calls to the **InstanceHandle** property are directed to the **InstanceHandle32** property.

IsChanged property

See also Example

Determines whether a master has changed since it was opened.

Version added

2002

Syntax

boolRet = *object*.**IsChanged**

boolRet	**Boolean**. **True** if the master has changed since it was opened; otherwise, **False** (0).
object	**Required**. An expression that returns a **Master** object.

IsConstant property

Example

Determines whether a formula of the cell is a constant expression.

Version added

4.0

Syntax

intRet = object.**IsConstant**

intRet	**Integer**. **True** (-1) if the object's formula is a constant; otherwise, **False** (0).
object	Required. An expression that returns a **Cell** object.

IsDefaultLink property

See also Example

Determines the default **Hyperlink** object for a shape.

Version added

2000

Syntax

boolVal = *object*.**IsDefaultLink**

object.**IsDefaultLink** = *boolExpression*

boolVal	**Boolean**. **True** if the **Hyperlink** object is the default; **False** if it isn't the default.
object	Required. An expression that returns a **Hyperlink** object.
boolExpression	Required **Boolean**. **True** to set the **Hyperlink** object as the default; otherwise, **False**.

Remarks

When you set the value of the **IsDefaultLink** property to **True** for a **Hyperlink** object, the value for all other **Hyperlink** objects is automatically set to **False**. When you set the value of this property to **False** for a **Hyperlink** object, the other **Hyperlink** objects aren't affected.

IsEditingOLE property

See also Example

Determines whether a drawing window contains an ActiveX control that has focus, or an embedded or linked object that is being edited.

Version added

2000

Syntax

boolRet = *object*.**IsEditingOLE**

boolRet	Required **Boolean**. **True** if an ActiveX control has focus or an OLE object is being edited; **False** if not, or if the window being examined is not a drawing window.
object	Required. An expression that returns a **Window** object.

IsEditingText property

Example

Determines whether a text editing session is active in the drawing window.

Version added

2000

Syntax

boolRet = *object*.**IsEditingText**

boolRet	Required **Boolean**. **True** if a text editing session is active; **False** if a text editing session is not active or the window being examined is not a drawing window.
object	Required. An expression that returns a **Window** object.

IsField property

Example

Determines whether a **Characters** object represents the expanded text of a single field with no additional non-field characters.

Version added

3.0

Syntax

intRet = object.**IsField**

intRet	**Integer**. **True** (-1) if the **Characters** object represents only the expanded text of a field; otherwise, **False** (0) if the **Characters** object contains characters in addition to the expanded text of a field.
object	Required. An expression that returns a **Characters** object.

Remarks

To change the range of text represented by a **Character** object, set its **Begin** and **End** properties.

IsHierarchical property

Indicates whether a menu, menu item, or toolbar item is hierarchical, that is, it contains a drop-down menu containing more items, which can be accessed through its own **MenuItems** or **ToolbarItems** collection menu.

Version added

4.0

Syntax

intRet = object.IsHierarchical

intRet	**Integer**. **True** (-1) if the object represents a hierarchical menu, menu item, or toolbar item; otherwise, **False** (0).
object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** object.

Remarks

The value of the **CmdNum** property of a **MenuItem** object that represents a hierarchical menu should be zero (0).

IsInherited property

Example

Determines whether a formula of the cell is inherited from a master or a style.

Version added

4.0

Syntax

```
intRet = object.IsInherited
```

intRet	**Integer**. **True** (-1) if the object's formula is inherited; otherwise, **False** (0).
object	Required. An expression that returns a **Cell** object.

Remarks

In the ShapeSheet window, the values and formulas of cells with local values appear in blue. Values and formulas of cells that inherit from a master or style appear in black.

IsInScope property

Determines whether a call to an event handler is between an **EnterScope** event and an **ExitScope** event for a scope.

Version added

2000

Syntax

boolVal = object.**IsInScope** (*nScopeID*)

boolVal	**Boolean**. **True** if an **EnterScope** event has fired for the scope ID, but an **ExitScope** event hasn't fired yet; **False** if an **EnterScope** event hasn't fired or **EnterScope** and **ExitScope** events have both fired.
object	Required. An expression that returns an **Application** object.
nScopeID	Required **Long**. The scope ID.

Remarks

Constants representing scope IDs are prefixed with **visCmd** and are declared by

the Visio type library. You can also use an ID returned by the **BeginUndoScope** method.

You could use this property in a **CellChanged** event handler to determine whether a cell change was the result of a particular operation.

IsOpenForTextEdit property

Example

Indicates whether a shape is currently open for interactive text editing.

Version added

2000

Syntax

boolRet = *object*.**IsOpenForTextEdit**

boolRet	Required **Boolean**. **True** if the shape is open for text editing in at least one window; **False** if the shape is not open for text editing.
object	Required. An expression that returns a **Shape** object.

IsSeparator property

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, the **IsSeparator** property represented a separator on a menu.

IsUndoingOrRedoing property

Determines whether the current event handler is being called as a result of an **Undo** or **Redo** action in the application.

Version added

2000

Syntax

boolRet = *object*.**IsUndoingOrRedoing**

boolRet	Required **Boolean**. **True** (-1) if the application is firing events related to an **Undo** or **Redo** action; otherwise, **False** (0).
object	Required. An expression that returns an **Application** object.

Remarks

The **IsUndoingOrRedoing** property returns **True** when the application is firing events related to an **Undo** or **Redo** action that the user has initiated through the user interface, or which an Automation client has initiated by calling the **Undo** or **Redo** method of an **Application** object.

When the application calls an event handler, the event has a "cause." If that cause is a user action or another event handler, then it is legitimate to perform undoable actions during the course of handling that event. However, if the cause of the event firing is an **Undo** or **Redo** action, then the event handler should not perform undoable actions. Doing so eliminates the ability to redo an action.

You will typically only perform undoable actions inside an event handler when this property is **False**. You can perform undoable actions when the flag is **True**, but the redo queue is destroyed.

Item[U] property

Returns an object from a collection. The **Item** property is the default property for all collections.

Version added

2.0

Syntax

objRet = *object*.**Item**(*nameUIDOrIndex*)

objRet	The object retrieved from the collection.
object	Required. An expression that returns a collection in the **Applies to** list.
nameUIDOrIndex	Required **Long**, **String**, or **Variant** (see Remarks for details). Contains the name, unique ID, or index of the object to retrieve.

Remarks

The data type for *nameUIDOrIndex* depends on the value of *object*.

Data type for *nameUIDOrIndex*	Values of *object*
Long	**AccelItems**, **AcceItables**, **Colors**, **Connects**, **MenuItems**, **Menu**, **Menusets**, **Path**, **Paths**, **Selection**, **ToolbarItems**, **Toolbars**, and **ToolbarSets**
String	**Eventlist**, **Windows**
Variant	**Addons**, **Documents**, **Fonts**, **Hyperlinks**, **Layers**, **Masters**, **MasterShortcuts**, **OLEObjects**, **Pages**, **Shapes**, and **Styles**

When retrieving objects from a collection, you can omit **Item** from the expression because it is the default property for all collections. The following statements are equivalent to the syntax example given above:

```
objRet = object(index)
objRet = object(stringExpression)
```

You can retrieve an object in a **Pages**, **Documents**, **Fonts**, **Layers**, **Masters**, **MasterShortcuts**, **Styles**, **Shapes**, **Addons**, or **OLEObjects** collection by passing the object's name as a string expression in a **Variant**.

If you retrieve a **Shape** object by name, the **Item** property searches all shapes in the **Shapes** collection's containing page or containing master, in addition to the collection's containing shape. Therefore, the **Shape** object returned by the **Item** property can be a shape that is not in the **Shapes** collection.

You can also pass the unique ID string of a **Master** or **Shape** object to the **Item** property. For example:

```
objRet = shpsObj.Item("{2287DC42-B167-11CE-88E9-0(
```

If such a string is passed to the **Item** property of a **Shapes** collection, all the shapes contained in the collection are searched. Shapes within the group shapes in the containing shape are not searched.

To search all shapes in the collection, plus the shapes inside groups and the containing shape of the collection, prefix the unique ID string with "*". For

example:

```
objRet = shpsObj.Item("*{2287DC42-B167-11CE-88E9-(
```

Note In Visio 2000 only, *shpsObj*.**Item**("{*guid*}") examined the **Shapes** collection's containing shape and all descendants, and *shpsObj*.**Item**("*{*guid*}") examined all shapes in the **Shapes** collection's containing page or containing master.

Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **Item** property to access an object in the **Masters**, **Pages**, **Shapes**, **Styles**, **Layers**, or **MasterShortcuts** collection using its local name. Use the **ItemU** property to access an object from one of these collections using the object's universal name.

ItemAtID property

Returns the **AccelTable**, **MenuSet**, or **ToolbarSet** object for an ID within a collection.

Version added

4.0

Syntax

objRet = *object*.**ItemAtID**(*id*)

objRet	The object retrieved from the collection.
object	Required. An expression that returns an **AccelTables**, **MenuSets**, or **ToolbarSets** collection.
id	Required **Long**. The Visio context ID of the object to retrieve.

Remarks

The ID corresponds to a window or context menu. Constants for IDs are prefixed with **visUIObjSet** and are declared by the Visio type library. For a list of valid IDs by collection, see the **SetID** property.

ItemFromID property

Example

Returns an item of a collection using the ID of the item.

Version added

4.0

Syntax

objRet = *object*.**ItemFromID**(*id*)

objRet	The object retrieved from the collection.
object	Required. An expression that returns a collection from the **Applies to** list.
id	Required **Long**. The ID of the object to retrieve.

Remarks

The ID of a **Shape** object uniquely identifies the shape within its page or master.

The ID of a **Style** object uniquely identifies the style within its document.

The ID of a **Font** object corresponds to the number stored in the Font cell of a row in a shape's Character Properties section. The ID associated with a particular font varies between systems or as fonts are installed on and removed from a given system.

The ID of an **Event** object uniquely identifies an event in its **EventList** collection for the life of the collection.

ItemStatus property

Example

Indicates if an item in a **Selection** object is subselected, if the group to which it belongs is selected, or if it is the primary item.

Version added

2000

Syntax

intRet = *object*.**ItemStatus**(*index*)

intRet	**Integer**. Status of the item.
object	Required. An expression that returns a **Selection** object.
index	Required **Long**. Index of the item for which you want to retrieve the status.

Remarks

The **ItemStatus** property reports a combination of the following values.

Constant	Value	Description

visSelIsPrimaryItem	&H1	The item is the primary item.
visSelIsSubItem	&H2	The item is a subselected item.
visSelIsSuperItem	&H4	The item is a superselected item.

IterationMode property

Example

Controls whether a **Selection** object reports subselected shapes and groups in which a shape is selected.

Version added

2000

Syntax

intRet = *object*.**IterationMode**

object.**IterationMode** = *intExpression*

intRet	**Long**. Mode of the selection.
object	Required. An expression that returns a **Selection** object.
intExpression	Required **Long**. Bit mask indicating whether sub- and super-selected items should be reported.

Remarks

The items in a **Selection** object are a subset of the descendants of the **Selection**

object's containing shape.

A top-level shape in a **Selection** object is an immediate child of the selection's containing shape.

A subselected shape in a **Selection** object is not an immediate child of the selection's containing shape.

A superselected shape in a **Selection** object has at least one immediate child that is subselected.

If a shape is subselected, then each of its ancestors—except the containing shape itself—is superselected.

The value of the **IterationMode** property is a combination of the following values.

Constant	Value	Description
visSelModeSkipSuper	&H0100	Selection does not report superselected shapes.
visSelModeOnlySuper	&H0200	Selection only reports superselected shapes.
visSelModeSkipSub	&H0400	Selection does not report subselected shapes.
visSelModeOnlySub	&H0800	Selection only reports subselected shapes.

When a **Selection** object is created, its initial iteration mode is **visSelModeSkipSub** + **visSelModeSkipSuper**. It reports neither subselected nor superselected shapes and behaves identically to **Selection** objects in versions of Visio prior to Visio 2000.

You can determine whether an individual item in a **Selection** object is a subselected or superselected item using the **ItemStatus** property.

Key property

Example

Gets or sets the ASCII key code value for an accelerator.

Version added

4.0

Syntax

```
keyVal = object.Key
object.Key = keyVal
```

keyVal	Required **Integer**. The ASCII value of the key used by the accelerator.
object	Required. An expression that returns an **AccelItem** object.

Remarks

For a list of ASCII key code values, search for "Virtual-Key Codes" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

Keywords property

Returns or sets the value of the **Keywords** box in a document's properties.

Version added

2.0

Syntax

strRet = *object*.**Keywords**

object.**Keywords** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The new value of the field.

Remarks

Setting the **Keywords** property is equivalent to entering information in the **Keywords** box in the **Properties** dialog box (click **Properties** on the **File** menu).

Language property

See also Example

Represents the language ID of the version of the Microsoft Visio instance represented by the **Application** object.

Version added

3.0

Syntax

longRet = *object*.**Language**

longRet	**Long**. The language ID.
object	Required. An expression that returns an **Application** object.

Remarks

The **Language** property returns the language ID recorded in the object's VERSIONINFO resource. The IDs returned are the standard IDs used by Microsoft Windows to encode different language versions. For example, the **Language** property returns &H0409 for the U.S. English version of Visio. For details, search for "VERSIONINFO" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

LargeButtons property

See also Example

Determines whether large toolbar buttons are shown.

Version added

2000

Syntax

boolVal = *object*.**LargeButtons**

object.**LargeButtons** = *boolExpression*

boolVal	**Boolean**. **True** (non-zero) if large buttons are shown; **False** (0) if large buttons are not shown.
object	Required. An expression that returns a **UIObject** object.
boolExpression	Required **Boolean**. **True** (non-zero) to show large buttons; **False** to show small buttons.

Remarks

You can use any **UIObject** object to get or set this property. The property affects the entire application, and affects the appearance of buttons in the *current visible set of toolbars*.

Beginning with Microsoft Visio 2002, this setting corresponds to the **Large icons** check box on the **Options** tab in the **Customize** dialog box (on the **Tools** menu, click **Customize**) and is shared between Visio 2002 and all Microsoft Office XP applications.

Layer property

Returns the layer to which a shape is assigned.

Version added

4.0

Syntax

objRet = *object*.**Layer**(*index*)

objRet	A **Layer** object that represents the requested layer.
object	Required. An expression that returns a **Shape** object.
index	**Integer**. The ordinal of the layer to get.

Remarks

If a shape is assigned to three layers, then the valid indexes that can be passed to its **Layer** property are 1 through 3.

To get the number of layers to which a shape is assigned, use the **LayerCount** property.

LayerCount property

Returns the number of layers to which a shape is assigned.

Version added

4.0

Syntax

intRet = *object*.**LayerCount**

intRet	**Integer**. The number of layers to which the shape is assigned.
object	Required. An expression that returns a **Shape** object.

Remarks

A shape is assigned to zero or more layers.

Layers property

Example

Returns the **Layers** collection of an object.

Version added

4.0

Syntax

objRet = *object*.**Layers**

objRet	The **Layers** collection of the **Master** or **Page** object.
object	Required. An expression that returns a **Master** or **Page** object.

Left property

Gets the distance between the left edge of the object and the left side of the docking area. Sets the distance between the left edge of a **Menu** or **Toolbar** object and the left edge of the screen.

Version added

2000

Syntax

intLong = *object*.**Left**

object.**Left** = *intLong*

object	Required. An expression that returns a **MenuSet** or **Toolbar** object.
intLong	Required **Integer**. Distance in pixels.

Remarks

The value of *intLong* must be greater than or equal to zero.

LeftMargin property

Example

Specifies the left margin, which is used when printing.

Version added

4.0

Syntax

retVal =object.**LeftMargin**([*unitsNameOrCode*])

object.**LeftMargin**([*unitsNameOrCode*]) = *newValue*

retVal	**Double**. The margin value expressed in the given units.
object	Required. An expression that returns a **Document** object.
unitsNameOrCode	Optional **Variant**. The units to use when retrieving or setting the margin value. Defaults to internal drawing units.
newValue	Required **Double**. The new margin value.

Remarks

The **LeftMargin** property corresponds to the **Left** setting in the **Print Setup**

dialog box (on the **File** menu, click **Page Setup**, click the **Print Setup** tab, and then click **Setup**).

You can specify *unitsNameOrCode* as an integer or a string value. If the string is invalid, an error is generated. For example, the following statements all set *unitsNameOrCode* to inches.

Cell.LeftMargin(**visInches**) = *newValue*

Cell.LeftMargin (65) = *newValue*

Cell.LeftMargin ("in") = *newValue* where "in" can also be any of the alternate strings representing inches, such as "inch", "in.", or "i".

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

LengthIU property

Example

Returns the length (perimeter) of the object in internal units.

Version added

4.0

Syntax

retVal = object.**LengthIU**

retVal	**Double**. The length (perimeter) of the object in internal units (inches).
object	Required. An expression that returns a **Shape** object.

LineBasedOn property

Example

Gets or sets the line style on which a **Style** object is based.

Version added

4.0

Syntax

strVal = *object*.**LineBasedOn**

object.**LineBasedOn** = *styleName*

strVal	**String**. The name of the current based-on line style.
object	Required. An expression that returns a **Style** object.
styleName	Required **String**. The name of the new based-on line style.

Remarks

To base a style on no style, set the **LineBasedOn** property to a zero-length string ("").

LineStyle property

Specifies the line style for an object.

Version added

2.0

Syntax

strRet = *object*.**LineStyle**

object.**LineStyle** = *stringExpression*

strRet	**String**. The name of the current line style.
object	Required. An expression that returns a **Shape** or **Selection** object.
stringExpression	Required **String**. The name of the line style to apply.

Remarks

Setting the **LineStyle** property is equivalent to selecting a line style from the **Line Style** list on the **Format Shape** toolbar in Visio.

Setting a style to a nonexistent style generates an error. Setting one kind of style to an existing style of another kind (for example, setting the **LineStyle** property to a fill style) does nothing. Setting one kind of style to an existing style that has more than one set of attributes changes only the attributes for that component. For example, setting the **LineStyle** property to a style with line, text, and fill attributes changes only the line attributes.

To preserve a shape's local formatting, use the **LineStyleKeepFmt** property.

Beginning with Microsoft Visio 2002, a zero-length string ("") will cause the master's style to be reapplied to the selection or shape. (Earlier versions generate a "no such style" exception.) If the selection or shape has no master, its style remains unchanged. Setting *stringExpression* to a zero-length string is the equivalent of selecting **Use master's format** in the **Text style**, **Line style**, or **Fill style** list in the **Style** dialog box (on the **Format** menu, click **Style**).

LineStyleKeepFmt property

Example

Applies a line style to an object while preserving local formatting.

Version added

2.0

Syntax

object.**LineStyleKeepFmt** = *stringExpression*

object	Required. An expression that returns a **Shape** or **Selection** object.
stringExpression	Required **String**. The name of the style to apply.

Remarks

Setting the **LineStyleKeepFmt** property is equivalent to selecting the **Preserve local formatting** check box in the **Style** dialog box (click **Style** on the **Format** menu).

Setting a style to a nonexistent style generates an error. Setting one kind of style

to an existing style of another kind (for example, setting the **LineStyleKeepFmt** property to a fill style) does nothing. Setting one kind of style to an existing style that has more than one set of attributes changes only the attributes for that component (for example, setting the **LineStyleKeepFmt** property to a style with line, text, and fill attributes changes only the line attributes).

Beginning with Microsoft Visio 2002, a zero-length string ("") will cause the master's style to be reapplied to the selection or shape. (Earlier versions generate a "no such style" exception.) If the selection or shape has no master, its style remains unchanged. Setting *stringExpression* to a zero-length string is the equivalent of selecting **Use master's format** in the **Text style**, **Line style**, or **Fill style** list in the **Style** dialog box (on the **Format** menu, click **Style**).

LiveDynamics property

Example

Controls whether Microsoft Visio recalculates shape properties during drag operations on every mouse move or only after the mouse button is released.

Version added

2000

Syntax

bLiveDynamics = *object*.**LiveDynamics**

object.**LiveDynamics** = *bLiveDynamics*

bLiveDynamics	Required **Boolean**. **True** if live dynamics is enabled; **False** if live dynamics is not enabled.
object	Required. An expression that returns an **Application** object.

Remarks

The **LiveDynamics** property tracks actions, such as resizing and rotating shapes, and is effective when shapes are glued or related to each other. When the value

of the **LiveDynamics** property is **True**, more events such as **CellChanged** occur. Solutions that respond to such events may operate more quickly if the **LiveDynamics** property is set to **False**.

LocalName property

Example

Returns the local name of a cell.

Version added

4.0

Syntax

strRet = *object*.**LocalName**

strRet	**String**. The local name of the cell.
object	Required. An expression that returns a **Cell** object.

Remarks

A cell has both a local name and a universal name. The local name differs according to the locale for which Microsoft Windows is installed on the user's system. The universal name is the same regardless of locale.

To get the universal name of a cell, use the **Name** property.

lParam property

See also Example

Gets or sets the **lParam** field of the **MSG** structure being wrapped.

Version added

2002

Syntax

intRet = *object*.**lParam**

object.**lParam**= *intValue*

intRet	**Long**. Additional information about the message that is dependent on the message number.
object	Required. An expression that returns a **MSGWrap** object.
intValue	Required **Long**. The new value of the **lParam** field.

Remarks

The **lParam** property corresponds to the **lParam** field in the **MSG** structure defined as part of the Microsoft Windows operating system. If an event handler is handling the **OnKeystrokeMessageForAddon** event, Visio passes a **MSGWrap** object as an argument when this event fires. A **MSGWrap** object is a wrapper around the Windows **MSG** structure.

For details, search for "MSG structure" on the Microsoft Developer Network (MSDN) Web site.

Manager property

Example

Returns or sets the value of the **Manager** box in a document's properties.

Version added

5.0

Syntax

strRet = *object*.**Manager**

object.**Manager** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The new value of the field.

Remarks

Setting the **Manager** property is equivalent to entering information in the **Manager** box in the **Properties** dialog box (click **Properties** on the **File** menu).

Master property

Example

Gets the master that is displayed in a window, returns the master from which the **Shape** object was created, or returns the master that contains the **Layer** object or **Layers** collection.

Version added

2.0

Syntax

objRet = *object*.**Master**

objRet	A **Master** object that represents the object's master.
object	Required. An expression that returns a **Layer** object, **Layers** collection, **Shape** object, or **Window** object.

Remarks

You can use the **Type** property of the **Window** object to determine whether the **Window** object shows a master. If the **Window** object does not show a master, the **Master** property raises an exception.

If the **Window** object shows a master that is open for editing, the master returned is the actual master being edited, not the temporary master that exists while the actual master is being edited.

If the **Shape** object is not an instance of a master, its **Master** property returns **Nothing**.

If the **Shape** object is in a group, its **Master** property is the same as the group's **Master** property.

If the **Layer** object or **Layers** collection is from a page rather than a master, its **Master** property returns **Nothing**.

Masters property

Returns the **Masters** collection for a document's stencil.

Version added

2.0

Syntax

objsRet = *object*.**Masters**

objsRet	The **Masters** collection for a document.
object	Required. An expression that returns a **Document** object.

MasterShape property

See also Example

Returns the shape in the master that this shape inherits from if this shape is part of a master instance.

Version added

2002

Syntax

objRet = *object*.**MasterShape**

objRet	The **Shape** object in the master that this object inherits from.
object	Required. An expression that returns a **Shape** object that is part of a master instance.

Remarks

Each shape in an instance of a master (the group and all of its subshapes) point to their corresponding shape in the master. The **MasterShape** property returns the **Shape** object in the master from which *object* inherits.

If *object* is not part of a master instance, the **MasterShape** property returns **Nothing**.

MasterShortcut property

See also Example

Gets the master shortcut that is displayed in a window.

Version added

2000

Syntax

objRet = *object*.**MasterShortcut**

objRet	A **MasterShortcut** object that is displayed in a window.
object	Required. An expression that returns a **Window** object.

MasterShortcuts property

See also

Returns the **MasterShortcuts** collection for a document stencil.

Version added

2000

Syntax

objsRet = *object*.**MasterShortcuts**

objsRet	The **MasterShortcuts** collection for a document.
object	Required. An expression that returns a **Document** object.

MatchByName property

See also Example

Determines how the application decides if a document master is already present when an instance of a master is dropped on the drawing page. It allows changes made to a document master to apply to new instances of the master, even if the instances are dragged from a stand-alone stencil file.

Version added

5.0

Syntax

intRet = *object*.**MatchByName**

object.**MatchByName** = *intExpression*

intRet	**Integer**. Non-zero if match by name is enabled; otherwise, zero (0).
object	Required. An expression that returns a **Master** object.
intExpression	Required **Integer**. Non-zero if match by name is enabled; otherwise, 0.

Remarks

Setting the **MatchByName** property is equivalent to selecting or clearing the **Match master name on drop** check box in the **Master Properties** dialog box (right-click the master, and then click **Master Properties** on the shortcut menu).

Suppose you create an instance of a master from a stencil in a document

(producing a local copy of the master in that document), and then make modifications to the document master (such as changing its fill color). If the **MatchByName** property of the document master is **False**, then dragging the original master from the stand-alone stencil into the drawing creates an instance with the stand-alone master's attributes and produces a second document master. If the **MatchByName** property of the document master is **True**, then dragging the original master from the stand-alone stencil into the drawing creates an instance with the document master's attributes and doesn't produce a second document master.

MDIWindowMenu property

See also Example

Determines whether this menu can be used by the MDI window manager to list the currently open MDI windows.

Version added

4.0

Syntax

intVal = *object*.**MDIWindowMenu**

object.**MDIWindowMenu** = *intVal*

intVal	Required **Integer**. Non-zero if the **Menu** object should be the MDI window menu; otherwise, zero (0).
object	Required. An expression that returns a **Menu** object.

Remarks

The **MDIWindowMenu** property usually refers to the **Window** menu.

MenuAnimationStyle property

See also Example

Gets or sets the way in which a menu is displayed.

Version added

2000

Syntax

int = *object*.**MenuAnimationStyle**

object.**MenuAnimationStyle** = *int*

object	Required. An expression that returns a **UIObject** object for which you want to set the menu animation style.
int	Required **Integer**. A constant that represents an animation style.

Remarks

You can use any **UIObject** object to get or set this property. The property affects the entire application, and affects the appearance of buttons in the *current visible set of toolbars*.

Beginning with Microsoft Visio 2002, this setting corresponds to the **Menu animations** box on the **Options** tab in the **Customize** dialog box (on the **Tools** menu, click **Customize**) and is shared between Visio 2002 and all Microsoft Office XP applications.

Constants representing animation styles are prefixed with **visMenuAnimation**

and are declared by the Visio type library in member **VisUIMenuAnimation**.

Constant	Value
visMenuAnimationNone	0
visMenuAnimationRandom	1
visMenuAnimationUnfold	2
visMenuAnimationSlide	3

MenuItems property

Returns the **MenuItems** collection of a **Menu** or **MenuItem** object.

Version added

4.0

Syntax

objRet = *object*.**MenuItems**

objRet	The **MenuItems** collection of the object.
object	Required. An expression that returns a **Menu** or **MenuItem** object that owns the collection.

Remarks

If a **Menu** object represents a hierarchical menu, its **MenuItems** collection contains submenu items. Otherwise, its **MenuItems** collection is empty.

Menus property

Returns the **Menus** collection of a **MenuSet** object.

Version added

4.0

Syntax

objRet = *object*.**Menus**

objRet	The **Menus** collection of the **MenuSet** object.
object	Required. An expression that returns a **MenuSet** object.

Remarks

A **Menu** object's index within the **Menus** collection determines its left-to-right position on the menu bar.

MenuSets property

Returns the **MenuSets** collection of a **UIObject** object.

Version added

4.0

Syntax

objRet = *object*.**MenuSets**

objRet	The **MenuSets** collection of a **UIObject** object.
object	Required. An expression that returns the **UIObject** object that owns the collection.

Remarks

If a **UIObject** object represents menus and accelerators (for example, if the object was retrieved using the **BuiltInMenus** property of an **Application** or **Document** object), its **MenuSets** collection represents all of the menus for that **UIObject** object.

Use the **ItemAtID** property of a **MenuSets** object to retrieve menus for a particular window context such as the drawing window. If a context does not include menus, it has no **MenuSets** collection.

MergeCaption property

Example

Returns the abbreviated caption that appears on the page tab when the window is merged with other windows.

Version Added

2002

Syntax

strRet = *object*.**MergeCaption**

object.**MergeCaption** = *strVal*

strRet	**String**. The text that appears in the page tab when this window is merged with other windows.
object	Required. An expression that returns a **Window** object.
strVal	Required **String**. The text to appear in the page tab when this window is merged with other windows.

Remarks

The **MergeCaption** property applies only to anchored windows. If the **Window** object is an MDI frame window, Visio raises an exception.

Use the **Type** property to determine window type.

MergeClass property

Example

Specifies a list of window classes that this anchored window can merge with.

Version Added

2002

Syntax

strRet = *object*.**MergeClass**

object.**MergeClass** = *strVal*

strRet	**String**. A list of window classes that this window can presently merge with.
object	Required. An expression that returns a **Window** object.
strVal	Required **String**. A new list of window classes that this window can merge with.

Remarks

Use semicolons to separate individual items in the list. If the **MergeClass** property returns a string containing "123;789", it can merge with any windows that also contain "123" or "789" in its merge class list. Windows with a merge class list that contains a zero-length string ("") can merge with other windows that contain a zero-length string ("") in their merge class list.

The **MergeClass** property applies only to anchored windows. If the **Window** object is an MDI frame window, Visio raises an exception.

At present, windows of type **visDocked** can be merged only with other windows of type **visDocked**, and windows of type **visAnchorBar** can be merged only with other windows of type **visAnchorBar**.

Use the **Type** property to determine window type.

MergeID property

Example

Specifies the string version of a merged window's globally unique identifier (GUID).

Version Added

2002

Syntax

strRet = *object*.**MergeID**
object.**MergeID** = *strVal*

strRet	**String**. The string version of a GUID.
object	Required. An expression that returns a **Window** object.
strVal	Required **String.** The new GUID**.**

Remarks

If this **Window** object is not merged, the GUID will contain all zeros (GUID_NULL).

The **MergeID** property applies only to anchored windows. If the **Window** object is an MDI frame window, Visio raises an exception.

Use the **Type** property to determine window type.

MergePosition property

Example

Specifies the left-to-right tab position of a merged anchored window.

Version Added

2002

Syntax

intRet = *object*.**MergePosition**

object.**MergePosition** = *intVal*

intRet	**Long**. The tab position of the merged window.
object	Required. An expression that returns a **Window** object.
intVal	Required **Long**. The new tab position of the merged window.

Remarks

If there are *n* tabs, the leftmost position is 1 and the rightmost position is *n*. If the windows are merged in a docked stencil fashion, 1 is the topmost and *n* is the

bottommost. A value of –1 means that the window is not merged.

The **MergePosition** property applies only to anchored windows. If the **Window** object is an MDI frame window, Visio raises an exception.

Use the **Type** property to determine window type.

message property

See also Example

Gets or sets the **message** field of the **MSG** structure being wrapped.

Version added

2002

Syntax

retVal = *object*.**message**

object.**message** = *newVal*

retVal	**Long**. The message identifier.
object	Required. An expression that returns a **MSGWrap** object.
newVal	Required **Long**. The new message identifier.

Remarks

The **message** property corresponds to the **message** field in the **MSG** structure defined as part of the Microsoft Windows operating system. If an event handler is handling the **OnKeystrokeMessageForAddon** event, Visio passes a **MSGWrap** object as an argument when this event fires. A **MSGWrap** object is a wrapper around the Windows **MSG** structure.

The **OnKeystrokeMessageForAddon** event fires for messages in the following range:

WM_KEYDOWN	0x0100

WM_KEYUP	0x0101
WM_CHAR	0x0102
WM_DEADCHAR	0x0103
WM_SYSKEYDOWN	0x0104
WM_SYSKEYUP	0x0105
WM_SYSCHAR	0x0106
WM_SYSDEADCHAR	0x0107

For details, search for "MSG structure" on the Microsoft Developer Network (MSDN) Web site.

Mode property

Example

Determines whether a document is in run mode or design mode.

Version added

5.0

Syntax

retVal = *object*.**Mode**

object.**Mode** = *newVal*

retVal	**VisDocModeArgs**. Current mode of the document.
object	Required. An expression that returns a **Document** object.
newVal	Required **VisDocModeArgs**. The new mode of the document; **visDocModeRun** (0) to set run mode, or **visDocModeDesign** (1) to set design mode.

Remarks

A Visio document is either in run mode or in design mode, just as a Microsoft

Visual Basic form is either running or being designed.

The following are the fundamental distinctions between run mode and design mode:

ActiveX controls hosted in a document are told not to fire events when the document is in design mode, and to fire events when in run mode.

Visio doesn't source events from any object whose document is in design mode.

The run/design mode of a Visio document is reported in the Visio user interface by the **Design Mode** button on the **Developer** toolbar. The appearance of this button is the same as the **Design Mode** button in the Visual Basic Editor window. If pressed, the document (project) is in design mode. If not pressed, the document (project) is in run mode.

The run/design mode of a Visio document is synchronized with the run/design state of the document's Visual Basic for Applications (VBA) project, provided the document has a project. If the document transitions to/from run mode, then the project's mode switches, and vice versa. This means that if code in a document's project sets the document's mode to design mode (**ThisDocument**.**Mode** = **visDocModeDesign**), the project in which the code executes transitions to design mode and any statements following the mode assignment statement don't execute. However, code in a document can put another document (project) into design mode and keep running.

A document's mode is not a persistent property. A document's initial mode is determined by the setting on the **Security Level** tab in the **Security** dialog box (on the **Tools** menu, point to **Macros**, and then click **Security**). The security levels are described as follows:

Low (not recommended), the document opens in run mode.

Medium, the document opens in run mode if it does not contain a project or contains a project from a trusted source. If the document contains a project that is unsigned or from an untrusted source, an alert appears with buttons to **Disable Macros** or **Enable Macros**. If **Enable Macros** is selected, the document opens in run mode; if **Disable Macros** is selected, it opens and remains in design mode.

High, the document opens in run mode if it does not contain a project or

contains a signed project from a trusted source. If the document contains an unsigned project, it opens and remains in design mode. If the document contains a signed project from an untrusted source, an alert appears with a **Disable** button and a dimmed **Enable** button (unless the **Always trust macros from this source** check box is selected). If **Disable** is selected the document opens and remains in design mode; otherwise, if **Enable** is selected, it opens in run mode.

Name[U] property

Specifies the name of an object.

Version added

2.0

Syntax

strRet = *object*.**Name**

object.**Name** = *stringExpression*

strRet	**String**. The current name of the object.
object	Required. An expression that returns a object from the **Applies to** list.
stringExpression	Required **String**. The new name of the object.

Remarks

You can get, but not set, the **Name** property of a **Document** object. If a document is not yet named, this property returns the document's temporary

name, such as Drawing1 or Stencil1.

You can get, but not set, the **Name** property of an **Addon** object or a **Font** object.

You can set the **Name** property of a **Style** object that represents a style that is not a default Visio style. If you attempt to set the **Name** property of a default Visio style, an error is generated.

You can get, but not set, the name of a cell. Some cells are in named rows; you can get and set the name of a named row using the **RowName** property.

A cell has both a local name and a universal name. The local name differs depending on the locale for which the running version of Microsoft Windows is installed. The universal name is the same regardless of what locale is installed. To get the universal name of a cell, use the **Name** property. To get the local name, use the **LocalName** property.

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **Name** property to get or set a **Cell**, **Master**, **Page**, **Shape**, **Style**, **Layer**, **Row**, or **MasterShortcut** object's local name. Use the **NameU** property to get or set its universal name.

NameID property

Example

Returns a unique name for a shape.

Version added

2.0

Syntax

```
strRet = object.NameID
```

strRet	**String**. The unique name of the shape.
object	Required. An expression that returns a **Shape** object.

Remarks

The **NameID** property returns a unique identifier for each shape on a page or master. The identifier has the following form: sheet.*N*, where *N* is the shape's **ID** property.

The value of the **NameID** property is unique within a page or master, but not across pages or masters. At any moment, no other shape on the same page or

master has the same **NameID** property. However, shapes on other pages or masters may have the same **NameID** property. The value of a shape's **UniqueID** property is unique across pages and masters.

Also, **NameID** properties are reused. If a shape whose **NameID** property is sheet.*N* is deleted, then a shape subsequently added to the same context may be assigned sheet.*N* as its **NameID** property. Therefore, persisting **NameID** properties in separate data stores is generally not as sound as persisting **UniqueID** properties.

NewBaseID property

Example

Generates a new base ID for a master.

Version added

2000

Syntax

strRet = *object*.**NewBaseID**

strRet	**String**. The new base ID for the master.
object	Required. An expression that returns a **Master** object.

NewWindow property

Determines whether Microsoft Visio opens a new window when it navigates to a URL.

Version added

5.0

Syntax

intRet = *object*.**NewWindow**

object.**NewWindow** = *intExpression*

intRet	**Integer**. Non-zero to open a new window; otherwise, zero (0).
object	Required. An expression that returns a **Hyperlink** object.
intExpression	Required **Integer**. Non-zero to open a new window; otherwise, zero (0).

Remarks

Setting the **NewWindow** property of a **Hyperlink** object is equivalent to setting

the NewWindow cell in the shape's Hyperlink.Row row.

Object property

Example

Returns an **IDispatch** interface on the ActiveX control or embedded or linked OLE 2.0 object represented by a **Shape** object or an **OLEObject** object.

Version added

4.1

Syntax

dispRet = *object*.**Object**

dispRet	**IDispatch** interface on the ActiveX control or OLE object represented by the shape.
object	Required. An expression that returns a **Shape** or **OLEObject** object.

Remarks

The **Object** property raises an exception if the object doesn't represent an ActiveX control or an OLE 2.0 embedded or linked object. A shape represents an ActiveX control or an OLE 2.0 embedded or linked object if the

visTypeIsOLE2 bit (&H8000) is set in the value returned by the **ForeignType** property.

If the **Object** property succeeds, it returns an **IDispatch** interface on the control or object. You owe an eventual release on the returned value (set it to **Nothing** or let it go out of scope if you're using Microsoft Visual Basic). You can determine the kind of object you've obtained an interface on by using the **ClassID** or **ProgID** property.

Beginning with Visio 5.0, if the object returned by the **Object** property is embedded and the shape inherits the object from its master, then the **Object** property severs the instance—that is, it copies the inherited data into the instance. Otherwise if the client receiving the **IDispatch** interface from the **Object** property makes changes to the object, all instances of the master, not just the instance being queried, change. If the object returned by the **Object** property is linked, the **Object** property does not sever the instance because, by definition, there may be other entities referencing the link. The **ObjectIsInherited** property was added to Visio 5.0 so that client programs can know if a shape inherits its object and access the master's object(s).

ObjectIsInherited property

Example

Indicates if a shape represents an ActiveX or OLE object that is inherited from the shape's master.

Version added

5.0

Syntax

intRet = object.**ObjectIsInherited**

intRet	**Integer**. **True** (-1) if object is inherited; otherwise, **False** (0).
object	Required. An expression that returns a **Shape** object.

ObjectType property

See also

Returns an object's type.

Version added

4.1

Syntax

intRet = *object*.**ObjectType**

intRet	**Integer**. The type of the object.
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

Constants representing object types are prefixed with **visObjType** and are declared by the Visio type library in **VisObjectTypes**.

- Constants representing object types

OLEObjects property

Returns the **OLEObjects** collection of a document, master, or page.

Version added

5.0

Syntax

objRet = *object*.**OLEObjects**

objRet	An **OLEObjects** collection.
object	Required. An expression that returns a **Document**, **Master**, or **Page** object.

Remarks

The **OLEObjects** property returns an **OLEObjects** collection that includes any OLE 2.0 linked or embedded objects, or ActiveX controls contained in a document, master, or page.

OnDataChangeDelay property

See also Example

Controls how long the Microsoft Visio instance waits before advising a container application that a Visio document being shown by the container has changed and should be redisplayed.

Version added

3.0

Syntax

intRet = *object*.**OnDataChangeDelay**

object.**OnDataChangeDelay** = *intExpression*

intRet	**Long**. The current setting of the object.
object	Required. An expression that returns an **Application** object.
intExpression	Required **Long**. The new setting of the object.

Remarks

The **OnDataChangeDelay** property only affects instances of Visio that are run from within an OLE container document.

Setting the value of the **OnDataChangeDelay** property to zero (0) causes Visio to send immediate advises to the container as data changes in open Visio documents.

Setting the value of the **OnDataChangeDelay** property to -1 causes Visio to use

the interval specified in the **OLEUpdateDelay** entry in the registry. If the registry doesn't contain this setting, Visio defaults to using a value of 10000 (milliseconds).

Setting the **OnDataChangeDelay** property to any value other than -1 or 0 overrides the registry setting and sets the delay between advises to the value of **OnDataChangeDelay**. If the **OnDataChangeDelay** property is not set or set to 1 and the **OLEUpdateDelay** setting is 0, Visio never sends advises to the container.

OneD property

See also Example

Determines whether an object behaves as a one-dimensional (1-D) object.

Version added

2.0

Syntax

retVal = *object*.**OneD**

object.**OneD** = *intExpression*

retVal	**Integer**. **True** if the shape is 1-D; **False** if the shape is 2-D.
object	Required. An expression that returns a **Master** or **Shape** object.
intExpression	Required **Integer**. Zero to declare object as 2-D; non-zero to declare it as 1-D.

Remarks

Setting the **OneD** property is equivalent to changing a shape's interaction style in the **Behavior** dialog box (click **Behavior** on the **Format** menu). Setting the **OneD** property for a 1-D shape to **False** deletes its 1-D Endpoints section, even if the cells in that section were protected with the GUARD function.

You can get, but not set, the **OneD** property of a **Master** object.

A guide does not have a **OneD** property.

The **OneD** property of an object from another application is always **False**.

Original property

See also Example

Returns the original master that produced this open master.

Version added

2002

Syntax

mastObj = *object*.**Original**

mastObj	The original **Master** object that produced this open master.
object	Required. An expression that returns a **Master** object.

Remarks

If this **Master** object is not an open copy of another **Master** object, then the **Original** property returns **Nothing**.

Page property (Layer object, Layers collection)

Gets the page that contains the layer.

Version added

2.0

Syntax

objRet = LayerOrLayersObj.**Page**

objRet	The **Page** object that contains the layer or layers.
LayerOrLayersObj	Required. An expression that returns a **Layer** object or **Layers** collection.

Remarks

If the **Layer** object or **Layers** collection is in a master rather than in a page, the **Page** property returns **Nothing**. You cannot set the **Page** property of a **Layer** object or **Layers** collection.

Page property (Window object)

See also Example

Gets or sets the page that is displayed in a window.

Version added

2.0

Syntax

objVariantRet = *windowObj*.**Page**

windowObj.**Page** = *stringVariant*

objVariantRet	**Variant**. A **Page** object that represents the page being shown returned in a **Variant**.
windowObj	Required. An expression that returns a **Window** object.
stringVariant	Required **Variant**. Contains a string that names the page to be shown.

Remarks

If a window is not showing a page (maybe it is showing a master), the **Page** property returns **Nothing**. You can use the **Type** property of the **Window** object to determine whether the **Window** object is showing a page. Otherwise, the returned **Variant** refers to the **Page** object that the window is showing.

Beginning with Visio 5.0b, the **Page** property no longer returns an exception if a window is not showing a page—it returns **Nothing**. You can use the following code to handle both return values.

```
'Close Window(i) if it is showing a page.
Set w = Windows(i)
On Error Resume Next
Set wp = w.Page
On Error GoTo 0
If Not wp Is Nothing Then
    w.Close
End If
```

Note In versions of Visio through version 4.1, the **Page** property of a **Window** object returned an **Object** (as opposed to a **Variant** of type **Object**) and the **Page** property of a **Window** object accepted a **String** (as opposed to a **Variant** of type **String**). Due to changes in Automation support tools, the property was changed to accept and return a **Variant**. For backward compatibility, the **PageAsObj** and **PageFromName** properties were added. The **PageAsObj** and **PageFromName** properties have the same signatures and occupy the same vtable slots as did the prior version of the **Page** property.

PageAsObj property

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property got the page that was displayed in a window.

PageFromName property

See also Example Applies to

Beginning with Microsoft Visio 2002, this property is obsolete.

Pages property

Returns the **Pages** collection for a document.

Version added

2.0

Syntax

objsRet = *object*.**Pages**

objsRet	The **Pages** collection for a document.
object	Required. An expression that returns a **Document** object.

PageSheet property

Example

Returns the page sheet of a page or master.

Version added

4.0

Syntax

objRet = *object*.**PageSheet**

objRet	A **Shape** object that represents a page sheet.
object	Required. An expression that returns a **Master** or **Page** object.

Remarks

Every page and master contains a tree of **Shape** objects. Constants representing shape types are prefixed with **visType** and are declared by the Visio type library.

In the tree of shapes of a master or page, there is exactly one shape of type **visTypePage**. This shape is always the root shape in the tree, and the **PageSheet** property returns this shape.

The page sheet contains important settings for the page or master such as its size and scale. It also contains the Layers section that defines the layers for that page or master.

An alternative way to obtain a page's or master's page shape is to use the following code:

```
shpObj = pageOrMasterObj.Shapes("ThePage")
```

PageTabWidth property

See also Example

Gets or sets the width of the page tab control in a drawing window.

Version added

2002

Syntax

dblRet = *object*.**PageTabWidth**

object.**PageTabWidth** = *dblVal*

dblRet	**Double**. The current width of the page tab control in the drawing window.
object	Required. An expression that returns a **Window** object.
dblVal	Required **Double**. The new width of the page tab control in the drawing window.

Remarks

The value in the **PageTabWidth** property is a percentage of the drawing window width, from 0 to 100. To use the default page tab control width, set the **PageTabWidth** property to –1.

PaletteEntry property

Example

Gets or sets the red, green, blue, and flags components of a color.

Version added

4.0

Syntax

intRet = *object*.**PaletteEntry**

object.**PaletteEntry** = *intVal*

intRet	**Long**. The current value of the color's components.
object	Required. An expression that returns a **Color** object.
intVal	Required **Long**. The new value of the color's components.

Remarks

A color is represented by 1-byte red, green, and blue components. It also has a 1-byte flags field indicating how you use the color. These correspond to members of the Windows **PALETTEENTRY** data structure. For details, search for

"**PALETTEENTRY**" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

The value passed is four tightly packed BYTE fields. The correspondence between the **PaletteEntry** property and red, green, blue, and flags values is:

palentry == r+256(b+256(g+256f))

PaletteWidth property

See also

Gets or sets the width of a palette in pixels.

Version added

2000

Syntax

intRet = *object*.**PaletteWidth**

object.**PaletteWidth** = *intValue*

intRet	**Integer**. Contains the width in pixels.
object	Required. An expression that returns an object in the **Applies to** list.
intValue	Required **Integer**. Contains the width in pixels.

Remarks

A palette, like a toolbar, is organized horizontally and items wrap to a new row if there is not enough horizontal space available. By default, only the icons of the

items are shown.

PaperHeight property

Example

Returns the height of a document's printed page.

Version added

4.5

Syntax

retVal = object.**PaperHeight**(*units*)

retVal	**Double**. The document's paper height expressed in the given units.
object	Required. An expression that returns a **Document** object.
units	Required **Variant**. The units to use when retrieving the paper height.

Remarks

Units can be a string such as "inches", "inch", "in.", or "i". Strings may be used for all supported Visio units such as centimeters, meters, miles, and so on. You can also use any of the units constants declared by the Visio type library in

member **VisUnitCodes**.

PaperSize property

Example

Gets or sets the paper size of a document.

Version added

4.5

Syntax

retVal = *object*.**PaperSize**

object.**PaperSize** = *newVal*

retVal	**VisPaperSizes**. The present page size.
object	Required. An expression that returns a **Document** object.
newVal	Required **VisPaperSizes**. The new page size.

Remarks

This is the equivalent of choosing a page size on the **Page Size** tab in the **Page Setup** dialog box (on the **File** menu, click **Page Setup**).The value of *retVal* and *newVal* can be one of the following **VisPaperSizes** constants.

Constant	Value	Description
visPaperSizeUnknown	0	Not known
visPaperSizeLetter	1	Letter 8 1/2 x 11 in
visPaperSizeLegal	5	Legal 8 1/2 x 14 in
visPaperSizeA3	8	A3 297 x 420 mm
visPaperSizeA4	9	A4 210 x 297 mm
visPaperSizeA5	11	A5 148 x 210 mm
visPaperSizeB4	12	B4 (JIS) 250 x 354 mm
visPaperSizeB5	13	B5 (JIS) 182 x 257 mm
visPaperSizeFolio	14	Folio 8 1/2 x 13 in
visPaperSizeNote	18	Note 8 1/2 x 11 in
visPaperSizeSizeC	24	C size sheet 17 x 22 in.
visPaperSizeSizeD	25	D size sheet 22 x 34 in.
visPaperSizeSizeE	26	E size sheet 34 x 44 in.

PaperWidth property

Example

Returns the width of a document's printed page.

Version added

4.5

Syntax

retVal = object.**PaperWidth**(*units*)

retVal	**Double**. The document's paper width expressed in the given units.
object	Required. An expression that returns a **Document** object.
units	Required **Variant**. The units to use when retrieving the paper width.

Remarks

Units can be a string such as "inches", "inch", "in.", or "i". Strings may be used for all supported Visio units such as centimeters, meters, miles, and so on. You can also use any of the units constants declared by the Visio type library.

ParaProps property

Example

Sets the paragraph property of a **Characters** object to a new value.

Version added

3.0

Syntax

object.**ParaProps**(*intWhichProp*) = *intExpressio*n

object	Required. An expression that returns a **Characters** object.
intWhichProp	Required **Integer**. The property to set.
intExpression	Required **Integer**. The new value of the property.

Remarks

The values of the *intWhichProp* argument correspond to named cells in the Paragraph section of the ShapeSheet window. Constants for *intWhichProp* are declared by the Visio type library in **VisCellIndices**.

Constant	Value

visIndentFirst	0
visIndentLeft	1
visIndentRight	2
visSpaceLine	3
visSpaceBefore	4
visSpaceAfter	5
visHorzAlign	6
visBulletIndex	7
visBulletString	8

Depending on the extent of the text range and the format, setting the **ParaProps** property may cause rows to be added or removed from the Paragraph section of the ShapeSheet window.

To retrieve information about an existing format, use the **ParaPropsRow** property.

ParaPropsRow property

Example

Returns the index of the row in the Paragraph section of a ShapeSheet window that contains paragraph formatting information for a **Characters** object.

Version added

3.0

Syntax

intRet = *object*.**ParaPropsRow**(*bias*)

intRet	**Integer**. The index of the row that defines the **Character** object's paragraph format.
object	Required. An expression that returns a **Characters** object.
bias	Required **Integer**. The direction of the search.

Remarks

If the formatting for the **Characters** object is represented by more than one row in the Paragraph section in the ShapeSheet window, the **ParaPropsRow** property returns -1. If the **Characters** object represents an insertion point rather

than a sequence of characters (its **Begin** and **End** properties return the same value), use the *bias* argument to determine which row index to return.

Constant	Value
visBiasLetVisioChoose	0
visBiasLeft	1
visBiasRight	2

Specify **visBiasLeft** for the row that covers paragraph formatting for the character to the left of the insertion point, or **visBiasRight** for the row that covers paragraph formatting for the character to the right of the insertion point.

Parent property

Example

Determines the parent of an object.

Version added

3.0

Syntax

objRet = *object*.**Parent**

objRet	The parent of the object.
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

In general, an object's parent is the object that contains it. For example, the parent of a **Menu** object is the **Menus** collection that contains the **Menu** object, or the parent of a **Window** object is the **Windows** collection that contains the **Window** object.

Parent property (Shape object)

Example

Determines the parent of a **Shape** object.

Version added

2002

Syntax

objRet = *object*.**Parent**

object.**Parent** = *objExpression*

objRet	The parent of the object.
object	Required. An expression that returns a **Shape** object.
objExpression	Required. The new parent of the object.

Remarks

In general, an object's parent is the object that contains it. If a **Shape** object is a member of a group, the parent is that group. Otherwise, its parent is a **Page** or a **Master** object.

When assigning a new parent shape, you must assign a **Shape** object. If you want to assign a page or master to be the parent of a shape, you must assign the **Shape** object returned by the **Page** or **Master** object's **PageSheet** property.

A shape and its parent shape must be in the same containing page or containing master. If the new parent is not a **Shape** object, or if the **ContainingPage** or **ContainingMaster** property of the parent shape is different from that of the shape, Visio raises an exception.

ParentItem property

See also Example

Returns the parent object of a hierarchical menu or toolbar.

Version added

2000

Syntax

objRet = *object*.**ParentItem**

objRet	The parent object.
object	Required. An expression that returns a **MenuItems** or **ToolbarItems** collection.

ParentWindow property

Example

Returns the **Window** object that is the parent of another **Window** object.

Version added

2000

Syntax

objRet = object.**ParentWindow**

objRet	The parent **Window** object.
object	Required. An expression that returns a **Window** object.

Remarks

ParentWindow returns nothing and raises no exception if the window is a top level window. A top level window is a member of the **Windows** collection of an **Application** object.

Use the **Parent** property of a **Window** object to get the **Windows** collection to which a **Window** object belongs.

Password property

Example

Resets the document's password.

Version added

2002

Syntax

object.**Password** ([*bStrExistingPassword*])= *strVal*

object	Required. An expression that returns a **Document** object.
bStrExistingPassword	Optional **Variant**. The existing password.
strVal	**String**. The new password.

Remarks

You can also set a document's password in the **Protect Document** dialog box (in the **Drawing Explorer**, right-click the drawing name, and then click **Protect Document**). If there is an existing password, you must first remove it by entering it in the **Unprotect Document** dialog box (in the **Drawing Explorer**, right-click the drawing name, and then click **Unprotect Document**).

Path property

Example

Returns the drive and folder path of the Microsoft Visio application or a document.

Version added

2.0

Syntax

strRet = *object*.**Path**

strRet	**String**. The path of Visio or the indicated document.
object	Required. An expression that returns an **Application** or **Document** object.

Remarks

If the document has not been saved, the **Path** property of the **Document** object returns a zero-length string ("").

Paths property

Returns a **Paths** collection that reports the coordinates of a shape's paths in the coordinate system of the shape's parent.

Version added

5.0

Syntax

objRet = *object*.**Paths**

objRet	A **Paths** object that represents the shape's strokes.
object	Required. An expression that returns a **Shape** object.

PathsLocal property

Example

Returns a **Paths** collection that reports the coordinates of a shape's paths in the shape's local coordinate system.

Version added

5.0

Syntax

objRet = object.**PathsLocal**

objRet	A **Paths** object that represents the shape's strokes.
object	Required. An expression that returns a **Shape** object.

PatternFlags property

See also Example

Determines whether a master behaves as a custom pattern.

Version added

5.0

Syntax

intRet = *object*.**PatternFlags**

object.**PatternFlags** = *intExpression*

intRet	**Integer**. The current value.
object	Required. An expression that returns a **Master** object.
intExpression	Required **Integer**. The new value.

Remarks

Visio allows a master to be used as a custom line pattern, line end, or fill pattern.

The **PatternFlags** property determines whether you can use a master as a pattern (non-zero); whether it is a line, fill, or line end pattern; and which pattern mode to use when applying it to shapes.

If you can use the **PatternFlags** property as a pattern (non-zero), the property can include a combination of the following bits.

Constant	Value	Description

visMasIsLinePat	&H1	Line pattern
visMasIsLineEnd	&H2	Line end pattern
visMasIsFillPat	&H4	Fill pattern

If **visMasIsLinePat** is selected, the pattern mode should be one of the following values.

Constant	Value
visMasLPTileDeform	&H0
visMasLPTile	&H10
visMasLPStretch	&H20
visMasLPAnnotate	&H30

In addition, **visMasLPScale** (&H40) can optionally be included in the **PatternFlag** property value.

If **visMasIsLineEnd** is selected, the pattern mode should be one of the following values.

Constant	Value
visMasLEDefault	&H0
visMasLEUpright	&H100

In addition, **visMasLEScale** (&H400) can optionally be included in the **PatternFlag** property value.

If **visMasIsFillPat** is selected, the pattern mode should be one of the following values.

Constant	Value
visMasFPTile	&H0
visMasFPCenter	&H1000
visMasFPStretch	&H2000

In addition, **visMasFPScale** (&H4000) can optionally be included in the **PatternFlag** property value.

Persistable property

Example

Determines whether an event can potentially persist within its document.

Version added

4.1

Syntax

intRet = *object*.**Persistable**

intRet	**Integer**. **False** (0) if the event cannot be made persistent; **True** (-1) if it can.
object	Required. An expression that returns an **Event** object.

Remarks

The **Persistable** property of an **Event** object indicates whether the event can persist, that is, whether the **Event** object can be stored with a Visio document between executions of a program. An **Event** object can persist if the following conditions are true:

The action code of the **Event** object must be **visActCodeRunAddon**. If the action code is **visActCodeAdvise**, the event won't persist and must be re-created by a program at run time.

The source object must be capable of containing persistent events in its **EventList** collection. The source object's **PersistsEvents** property indicates whether it can contain persistent events. The only source objects currently capable of containing persistent events are **Document**, **Master**, and **Page** objects.

If these conditions are met, an **Event** object with any of the following event codes is persistable:

DocumentCreated

DocumentOpened

MasterAdded

MasterDeleted

PageAdded

PageDeleted

ShapesDeleted

Although an **Event** object's **Persistable** property indicates whether an event can persist, its **Persistent** property indicates whether that event actually persists. When an **Event** object is first created, its **Persistent** property is set to the same value as its **Persistable** property. That is, a persistable event's **Persistent** property is set to **True**, and a nonpersistable event's **Persistent** property is set to **False**.

A nonpersistent event exists as long as a reference is held on the **Event** object, the **EventList** object that contains the **Event** object, or the source object that has the **EventList** object. When the last reference to any of these objects is released, the nonpersistent event ceases to exist.

You can change the initial setting for a persistable event by setting its **Persistent** property to **False**. In this case, the event doesn't persist with its document, even though it could. However, you cannot change the **Persistent** property of a

nonpersistent event; attempting to do so will cause an exception.

Note Events handled in a Microsoft Visual Basic for Applications project are persistent.

Persistent property

Example

Determines whether or not an event persists with its document.

Version added

4.1

Syntax

intRet = *object*.**Persistent**

object.**Persistent** = *intExpression*

intRet	**Integer**. **False** (0) if the event won't be saved with the document; **True** (-1) if it will.
object	Required. An expression that returns an **Event** object.
intExpression	Required **Integer**. **False** (0) to make the event nonpersistent; **True** (non-zero) to make it persistent.

Remarks

An event is persistable if its action code is **visActCodeRunAddon** and the

event's source object is capable of containing persistent events.

When an event is first created, its **Persistent** property is set to the same value as its **Persistable** property; if an event can persist, Visio assumes it should persist. You can change the initial setting for a persistable event by setting its **Persistent** property to **False**. However, you cannot change the **Persistent** property of a nonpersistable event—attempting to do so causes an exception.

A nonpersistent event exists as long as a reference is held on the **Event** object, the **EventList** object that contains the **Event** object, or the source object that has the **EventList** object. When the last reference to any of these objects is released, the nonpersistent event ceases to exist.

A persistent event exists until its **Event** object is deleted from the source object's **EventList** collection.

Note Events handled in a Microsoft Visual Basic for Applications project are persistent.

PersistsEvents property

Indicates whether an object is capable of containing persistent events in its **EventList** collection.

Version added

4.1

Syntax

intRet = object.**PersistsEvents**

intRet	**Integer**. **False** (0) if this object cannot contain persistent events; **True** (1) if it can.
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

Every object that has an **EventList** property also has a **PersistsEvents** property. To be persistable, an event's action code must be **visActCodeRunAddon**, but it must also be in the **EventList** collection of an object whose **PersistsEvents**

property is **True**. The only objects that currently persist events are **Document**, **Master**, and **Page** objects.

Whether a persistable event actually does persist depends on the value of its **Persistent** property.

Picture property

Example

Returns a picture that represents an enhanced metafile (EMF) contained in a master, shape, selection or page.

Version Added

2002

Syntax

objRet = *object*.**Picture**

objRet	An IPictureDisp object that represents the enhanced metafile.
object	Required. An expression that returns a **Master**, **Shape**, **Selection**, or **Page** object that contains the picture.

Remarks

The **Picture** property returns only EMF files (enhanced metafiles).

COM provides a standard implementation of a picture object with the **IPictureDisp** interface on top of the underlying system picture support. The

IPictureDisp interface exposes a picture object's properties and is implemented in the stdole type library as a **StdPicture** object creatable within Microsoft Visual Basic. The stdole type library is automatically referenced from all Visual Basic for Applications projects in Visio.

To get information about the **StdPicture** object that supports the **IPictureDisp** interface:

On the **Tools** menu, point to **Macros**, and then click **Visual Basic Editor**.

On the **View** menu, click **Object Browser**.

In the **Project/Library** list, click **stdole**.

Under **Classes**, examine the class named **StdPicture**.

For details about the **IPictureDisp** interface, see the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

Currently, only in-proc solutions can use the **Picture** property because the **IPictureDisp** interface cannot be marshaled.

PitchAndFamily property

Example

Returns the pitch and family code for a **Font** object.

Version added

4.0

Syntax

intRet = object.**PitchAndFamily**

intRet	**Integer**. The pitch and family code of the **Font** object.
object	Required. An expression that returns a **Font** object.

Remarks

Use the **PitchAndFamily** property to specify a font's pitch and assign it to a font family. You can specify pitch, family, or both. To specify both, use an or expression. Font families are used to specify a font when an exact typeface is unavailable.

The possible values of the **PitchAndFamily** property correspond to those of the

lfPitchAndFamily member of the Windows **LOGFONT** data structure. For details, search for "LOGFONT" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

Points property

Returns an array of points that defines a polyline that approximates a **Path** or **Curve** object within a given tolerance.

Version added

5.0

Syntax

object.**Points** *Tolerance*, *xyArray*

object	Required. An expression that returns a **Path** or **Curve** object.
Tolerance	Required **Double**. Specifies how close the returned array of points must approximate the true path.
xyArray	Required **Double**. Returns an array of alternating *x* and *y* values specifying points along a path's or curve's stroke.

Remarks

Use the **Points** property of the **Path** or **Curve** object to obtain an array of *x*,*y* coordinates specifying points along the path or curve within a given tolerance.

The tolerance and returned *x,y* values are expressed in internal drawing units (inches).

If you used the **Paths** property of a **Shapes** object to obtain the **Path** or **Curve** object being queried, the coordinates are expressed in the parent's coordinate system. If you used the **PathsLocal** property of a **Shape** object to obtain the **Path** or **Curv**e object, the coordinates are expressed in the local coordinate system.

If Visio is unable to achieve the requested tolerance, Visio approximates the points as close to the requested tolerance as possible. Generally speaking, the lower the tolerance, the more points Visio returns. Visio doesn't accept a tolerance of zero (0).

The array returned includes both the starting and ending points of the path or curve even if it is closed.

Position property

Gets or sets the position of an object.

Version added

2000

Syntax

intRet = *object*.**Position**

object.**Position** = *intExpression*

intRet	**Integer**. The object's position.
object	Required. An expression that returns a **MenuSet** or **Toolbar** object.
intExpression	Required **Integer**. The object's new position.

Remarks

Constants representing possible **Position** property values are listed below. They are also declared by the Visio type library in **VisUIBarPosition**.

Constant	Value
visBarLeft	0
visBarTop	1
visBarRight	2
visBarBottom	3
visBarFloating	4
visBarPopup	5
visBarMenu	6

posttime property

Example

Gets or sets the **time** field of the **MSG** structure being wrapped.

Version added

2002

Syntax

intRet = *object*.**posttime**

object.**posttime** = *intExpression*

intRet	**Long**. The current value of the **time** field.
object	Required. An expression that returns a **MSGWrap** object.
intExpression	Required **Long**. The new value of the **time** field.

Remarks

The **posttime** property corresponds to the **time** field in the **MSG** structure defined as part of the Microsoft Windows operating system. If an event handler is handling the **OnKeystrokeMessageForAddon** event, Visio passes a

MSGWrap object as an argument when this event fires. A **MSGWrap** object is a wrapper around the Windows **MSG** structure.

For details, search for "MSG structure" on the Microsoft Developer Network (MSDN) Web site.

PreviewPicture property

See also Example

Gets or sets the preview picture shown in the **Open** or **Choose Drawing Type** dialog box.

Version added

2002

Syntax

objRet = *object*.**PreviewPicture**

object.**PreviewPicture** = *objExpression*

objRet	An **IPictureDisp** object that represents current preview picture.
object	Required. An expression that returns a **Document** object.
objExpression	Required. An **IPictureDisp** object that represents the new preview picture.

Remarks

The **PreviewPicture** property returns and accepts only EMF files (enhanced metafiles). Visio will raise an exception if *objExpression* contains a non-EMF file.

To delete an existing preview set the **PreviewPicture** property to **Nothing**.

You can use the **PreviewPicture** property to include a preview pictures in a template that does not have any diagrams stored in it.

COM provides a standard implementation of a picture object with the **IPictureDisp** interface on top of the underlying system picture support. The **IPictureDisp** interface exposes a picture object's properties and is implemented in the stdole type library as a **StdPicture** object creatable within Microsoft Visual Basic. The stdole type library is automatically referenced from all Visual Basic for Applications projects in Visio.

To get information about the **StdPicture** object that supports the **IPictureDisp** interface:

On the **Tools** menu, point to **Macros**, and then click **Visual Basic Editor**.

On the **View** menu, click **Object Browser**.

In the **Project/Library** list, click **stdole**.

Under **Classes**, examine the class named **StdPicture**.

For details about the **IPictureDisp** interface, see the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

Currently, only in-proc solutions can use the **PreviewPicture** property because the **IPictureDisp** interface cannot be marshaled.

PrimaryItem property

Example

Returns the **Shape** object that is a **Selection** object's primary item.

Version added

2000

Syntax

objRet = object.**PrimaryItem**

objRet	The **Shape** object that is the primary item.
object	Required. An expression that returns a **Selection** object.

Remarks

In a drawing window, the primary selected item is shown with green selection handles and non-primary selected items are shown with blue selection handles. The outcome of some operations is affected by which selected item is the primary item. For example, the **Align Shapes** command aligns non-primary selected items with the primary selected item.

If a **Selection** object contains no **Shape** objects, or the primary **Shape** object is one that isn't enumerated given the **Selection** object's **IterationMode** property, the **PrimaryItem** property returns **Nothing** and raises no exception.

PrintCenteredH property

Example

Indicates whether drawings are centered between the left and right edges of the paper when printed.

Version added

4.0

Syntax

boolRet = *object*.**PrintCenteredH**

object.**PrintCenteredH** = *boolValue*

boolRet	**Boolean**. **True** if the document will center drawings horizontally when printing; otherwise, **False**.
object	Required. An expression that returns a **Document** object.
boolValue	Required **Boolean**. **True** to center drawings horizontally when printing; otherwise, **False**.

Remarks

The **PrintCenteredH** property corresponds to the **Center horizontally** check box in the **Print Setup** dialog box (click **Page Setup** on the **File** menu, and then click **Setup** on the **Print Setup** tab).

PrintCenteredV property

Example

Indicates whether drawings are centered between the top and bottom edges of the paper when printed.

Version added

4.0

Syntax

boolRet = *object*.**PrintCenteredV**

object.**PrintCenteredV** = *boolValue*

boolRet	**Boolean**. **True** if the document will center drawings vertically when printing; otherwise, **False**.
object	Required. An expression that returns a **Document** object.
boolValue	Required **Boolean**. **True** to center drawings vertically; otherwise, **False**.

Remarks

The **PrintCenteredV** property corresponds to the **Center vertically** check box in the **Print Setup** dialog box (click **Page Setup** on the **File** menu, and then click **Setup** on the **Print Setup** tab).

Printer property

Example

Specifies the name of the printer to use when printing the document.

Version added

2002

Syntax

strRet = *object*.**Printer**

object.**Printer** = *strExpression*

strRet	**String**. The current printer.
object	Required. An expression that returns a **Document** object.
strExpression	Required **String**. The new printer.

Remarks

The **Printer** property corresponds to the **Name** box in the **Print** dialog box (click **Print** on the **File** menu).

PrintFitOnPages property

Example

Indicates whether drawings in a document are printed on a specified number of sheets across and down.

Version added

4.0

Syntax

boolRet = *object*.**PrintFitOnPages**

object.**PrintFitOnPages** = *boolValue*

boolRet	**Boolean**. **True** if the document will fit drawings on a specified number of sheets; otherwise, **False**.
object	Required. An expression that returns a **Document** object.
boolValue	Required **Boolean**. **True** to fit drawings on a specified number of sheets; otherwise, **False**.

Remarks

The **PrintFitOnPages** property corresponds to the **Fit to** settings in the **Page Setup** dialog box (click **Page Setup** on the **File** menu). If this property is **True**, Visio prints the document's drawings on the number of sheets specified by the **PrintPagesAcross** and **PrintPagesDown** properties.

PrintLandscape property

See also Example

Indicates whether a document's drawings print in landscape or portrait orientation.

Version added

4.0

Syntax

boolRet = *object*.**PrintLandscape**

object.**PrintLandscape** = *boolValue*

boolRet	**Boolean**. **True** if the document will print drawings in landscape orientation; otherwise, **False**.
object	Required. An expression that returns a **Document** object.
boolValue	Required **Boolean**. **True** to print drawings in landscape orientation; otherwise, **False**.

Remarks

The **PrintLandscape** property corresponds to the **Portrait** and **Landscape** settings in the **Print Setup** dialog box (click **Page Setup** on the **File** menu, and then click **Setup**).

PrintPagesAcross property

Example

Indicates the number of sheets of paper on which a drawing is printed horizontally.

Version added

4.0

Syntax

intRet = *object*.**PrintPagesAcross**

object.**PrintPagesAcross** = *newValue*

intRet	**Integer**. The current number of pages across on which drawings are printed.
object	Required. An expression that returns a **Document** object.
newValue	Required **Integer**. The new number of pages across on which to print drawings.

Remarks

You must set the value of the **PrintFitOnPages** property to **True** in order to use the **PrintPagesAcross** property. If the value of the **PrintFitOnPages** property is **False**, Visio ignores the **PrintPagesAcross** property.

The **PrintPagesAcross** property corresponds to the **Fit to sheet(s) across** setting in the **Page Setup** dialog box (click **Page Setup** on the **File** menu).

PrintPagesDown property

Example

Indicates the number of sheets of paper on which a drawing is printed vertically.

Version added

4.0

Syntax

intRet = *object*.**PrintPagesDown**

object.**PrintPagesDown** = *newValue*

intRet	**Integer**. The number of sheets on which drawings are printed vertically.
object	Required. An expression that returns a **Document** object.
newValue	Required **Integer**. The number of sheets on which to print drawings vertically.

Remarks

You must set the value of the **PrintFitOnPages** property to **True** to use the

PrintPagesDown property. If the value of the **PrintFitOnPages** property is **False**, Visio ignores the **PrintPagesDown** property.

The **PrintPagesDown** property corresponds to the **Fit to by sheets down** setting in the **Page Setup** dialog box (click **Page Setup** on the **File** menu).

PrintScale property

Example

Indicates how much drawings are reduced or enlarged when printed.

Version added

4.0

Syntax

retVal = *object*.**PrintScale**

object.**PrintScale** = *newValue*

retVal	**Double**. The scale at which drawings are printed; 1.0 equals 100%.
object	Required. An expression that returns a **Document** object.
newValue	Required **Double**. The new scale value.

Remarks

The **PrintScale** property corresponds to the **Adjust to** setting on the **Print Setup** tab in the **Page Setup** dialog box (click **Page Setup** on the **File** menu). To print a

drawing at half its size, specify 0.5. To print a drawing at twice its size, specify 2.0.

PrintTileCount property

Example

Returns the number of print tiles for a drawing page.

Version added

2002

Syntax

```
intRet = object.PrintTileCount
```

intRet	**Long**. The number of print tiles on the drawing page.
object	Required. An expression that returns a **Page** object.

Remarks

When drawings span multiple physical printer pages, you can use the **PrintTileCount** property to determine the number of print tiles there are for a Visio drawing page. You can use the **PrintTileCount** property with the **PrintTile** method to identify and print selected tiles of an active drawing page.

Priority property

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property determined whether a toolbar or status bar item was dropped from view when the Visio window was too narrow to show all items.

ProcessID property

See also

Returns a unique process ID for a Microsoft Visio instance.

Version added

2.0

Syntax

retVal = object.**ProcessID**

retVal	**Long**. The process ID for the Visio instance.
object	Required. An expression that returns an **Application** object.

Remarks

The **ProcessID** property returns a value unique to the indicated instance. The application doesn't reuse the value until 4294967296 (2^32) more processes have been created on the current workstation.

ProductName property

See also Example Applies to

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property returned the name of the product, Visio, using its key in the registry.

ProfileName property

See also Example Applies to

Beginning with Microsoft Visio 2002, this method is obsolete.

Remarks

In Visio versions earlier than 2002, the **ProfileName** property returned the path to the Microsoft Visio information stored in the registry.

ProgID property

Returns the programmatic identifier of a shape representing an ActiveX control, an embedded object, or linked object.

Version added

5.0

Syntax

strRet = *object*.**ProgID**

strRet	**String**. The program identifier of the OLE object represented by the shape.
object	Required. An expression that returns a **Shape** object.

Remarks

The **ProgID** property raises an exception if the shape doesn't represent an ActiveX control or OLE 2.0 embedded or linked object. A shape represents an ActiveX control, an embedded object, or linked object if the **ForeignType** property returns **visTypeIsOLE2** in the value.

Use the **ProgID** property of a **Shape** object or **OLEObject** to obtain the programmatic identifier of the object. Every OLE object class stores a programmatic identifier for itself in the registry. Typically this occurs when the program that services the object installs itself. Client programs use this identifier to identify the object. You are using the Visio identifier when you execute a statement such as **GetObject**(,"Visio.Application") from a Microsoft Visual Basic program.

These are strings that the **ProgID** property might return:

Visio.Drawing.5
MSGraph.Chart.5
Forms.CommandButton.1

After using a shape's **Object** property to obtain an **IDispatch** interface on the object the shape represents, you can obtain the shape's **ClassID** or **ProgID** property to determine the methods and properties provided by that interface.

Prompt property

See also Example

Gets or sets the prompt string for a master or master shortcut.

Version added

2.0

Syntax

strRet = *object*.**Prompt**

object.**Prompt** = *strVal*

strRet	**String**. The current prompt string.
object	Required. An expression that returns a **Master** or **MasterShortcut** object.
strVal	**String**. The new prompt string.

PromptForSummary property

Determines whether Microsoft Visio prompts for document properties when it saves a document.

Version added

4.0

Syntax

intRet = *object*.**PromptForSummary**

object.**PromptForSummary** = *intExpression*

intRet	**Integer**. Zero (0) if prompting is off; -1 if it is on.
object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. Zero (0) to turn prompting off; non-zero to turn it on.

Remarks

This property corresponds to the **Prompt for document properties** check box

on the **Save** tab in the **Options** dialog box (click **Options** on the **Tools** menu).

Protection property

See also

Determines how an object is protected from user customization.

Version added

2000

Syntax

intRet = *object*.**Protection**

object.**Protection** = *intExpression*

intRet	**Integer**. The existing protections for a **MenuSet** or **Toolbar** object.
object	Required. An expression that returns a **MenuSet** or **Toolbar** object.
intExpression	Required **Integer**. The new protections for a **MenuSet** or **Toolbar** object.

Remarks

The value of *intExpression* can be one or a combination of the following constants declared by the Visio type library in **VisUIBarProtection**.

Constant	Value	Description
visBarNoProtection	0	No protection.
visBarNoCustomize	1	Can't be customized.
visBarNoResize	2	Can't be resized.
visBarNoMove	4	Can't be moved.
visBarNoChangeDock	16	Can't be docked or floating.
visBarNoVerticalDock	32	Can't be docked vertically.
visBarNoHorizontalDock	64	Can't be docked horizontally.

Protection property (Document object)

Example

Determines how a document is protected from user customization.

Version added

2002

Syntax

retVal = object.**Protection** ([*bstrPassword*])

object.**Protection** ([*bstrPassword*]) = *newValue*

retVal	**VisProtection**. The current protection settings.
object	Required. An expression that returns a **Document** object.
bstrPassword	Optional **Variant**. The existing password.
newValue	Required **VisProtection**. The new protection settings.

Remarks

If the document is password protected, you must provide the existing password to set the **Protection** property. If you provide an incorrect password, the **Protection** property will raise an exception.

The **Protection** property ignores *bstrPassword* when you are getting the value of the property.

This property is the equivalent of checking the **Styles**, **Shape**, **Preview**, **Backgrounds**, and **Master shapes** boxes on the **Protect Document** dialog box (in the **Drawing Explorer**, right-click the drawing name, and then click **Protect Document**). If there is an existing password, you must first remove it by entering it in the **Unprotect Document** dialog box (in the **Drawing Explorer**, right-click the drawing name, and click **Unprotect Document**).

The value of *retVal* and *newVal* can be a combination of the following **VisProtection** constants.

Constant	Value
visProtectNone	&H0
visProtectStyles	&H1
visProtectShapes	&H2
visProtectMasters	&H4
visProtectBackgrounds	&H8
visProtectPreviews	&H10

ptx property

See also Example

Gets or sets the **pt.x** field of the **MSG** structure being wrapped.

Version added

2002

Syntax

intRet = *object*.**ptx**

object.**ptx** = *intExpression*

intRet	**Long**. The *x*-coordinate of the cursor position when this message was posted.
object	Required. An expression that returns a **MSGWrap** object.
intExpression	Required **Long**. The new value of the **pt.x** field.

Remarks

The **ptx** property corresponds to the **pt.x** field in the **MSG** structure defined as part of the Microsoft Windows operating system. If an event handler is handling the **OnKeystrokeMessageForAddon** event, Visio passes a **MSGWrap** object as an argument when this event fires. A **MSGWrap** object is a wrapper around the Windows **MSG** structure.

For details, search for "MSG structure" on the Microsoft Developer Network (MSDN) Web site.

pty property

See also Example

Gets or sets the **pt.y** field of the **MSG** structure being wrapped.

Version added

2002

Syntax

intRet = *object*.**pty**

object.**pty** = *intExpression*

intRet	**Long**. The *y*-coordinate of the cursor position when this message was posted.
object	Required. An expression that returns a **MSGWrap** object.
intExpression	Required **Long**. The new value of the **pt.y** field.

Remarks

The **pty** property corresponds to the **pt.y** field in the **MSG** structure defined as part of the Microsoft Windows operating system. If an event handler is handling the **OnKeystrokeMessageForAddon** event, Visio passes a **MSGWrap** object as an argument when this event fires. A **MSGWrap** object is a wrapper around the Windows **MSG** structure.

For details, search for "MSG structure" on the Microsoft Developer Network (MSDN) Web site.

ReadOnly property

See also Example

Indicates whether a file is open as read-only.

Version added

2.0

Syntax

intRet = *object*.**ReadOnly**

intRet	**Integer**. Non-zero (**True**) if the document is open as read-only; otherwise, 0 (**False**).
object	Required. An expression that returns a **Document** object.

Red property

Example

Gets or sets the intensity of the red component of a **Color** object.

Version added

4.0

Syntax

intRet = *object*.**Red**

object.**Red** = *intVal*

intRet	**Integer**. The current value of the color's red component.
object	Required. An expression that returns a **Color** object.
intVal	Required **Integer**. The new value of the color's red component.

Remarks

The **Red** property can be a value from zero (0) to 255.

A color is represented by red, green, and blue components. It also has flags that

indicate how the color is to be used. These correspond to members of the Microsoft Windows **PALETTEENTRY** data structure. For details, search for "PALETTEENTRY" in the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

Result property

Gets or sets a cell's value.

Version added

2.0

Syntax

retVal = object.**Result** (*unitsNameOrCode*)

object.**Result** (*unitsNameOrCode*) = *newValue*

retVal	**Double**. The value in the cell.
object	Required. An expression that returns a **Cell** object.
unitsNameOrCode	Required **Variant**. The units to use when retrieving or setting the cell's value.
newValue	Required **Double**. The new value for the cell.

Remarks

Use the **Result** property to set the value of an unguarded cell. If the cell's

formula is protected with the GUARD function, the formula is not changed and an error is generated. If the cell contains only a text string, then zero (0) is returned. If the string is invalid, an error is generated.

You can specify *unitsNameOrCode* as an integer or a string value. For example, the following statements all set *unitsNameOrCode* to inches.

retVal = **Cell.Result**(**visInches**)

retVal = **Cell.Result**(65)

retVal = **Cell.Result**("in") where "in" can also be any of the alternate strings representing inches, such as "inch", "in.", or "i".

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

To specify internal units, pass a zero-length string (""). Internal units are inches for distance and radians for angles. To specify implicit units, you must use the **Formula** property.

ResultForce property

Example

Sets a cell's value, even if the cell's formula is protected with the GUARD function.

Version added

2.0

Syntax

object.**ResultForce** (*unitsNameOrCode*) = *newValue*

object	Required. An expression that returns a **Cell** object.
unitsNameOrCode	Required **Variant**. The units to use when setting the cell's value.
newValue	Required **Double**. The new value for the cell.

Remarks

Use the **ResultForce** method to set a cell's value even if the cell's formula is protected with a guard function. If the string is invalid, an error is generated.

You can specify *unitsNameOrCode* as an integer or a string value. For example, the following statements all set *unitsNameOrCode* to inches.

Cell.ResultForce(**visInches**) = *newValue*

Cell.ResultForce(65) = *newValue*

Cell.ResultForce("in") = *newValue* where "in" can also be any of the alternate strings representing inches, such as "inch", "in.", or "i".

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

To specify internal units, pass a zero-length string (""). Internal units are inches for distance and radians for angles. To specify implicit units, you must use the **Formula** property.

ResultFromInt property

Example

Sets the value of a cell to an integer value.

Version added

4.5

Syntax

object.**ResultFromInt**(*unitsNameOrCode*) = *newValue*

object	Required. An expression that returns a **Cell** object.
unitsNameOrCode	Required **Variant**. The units to use when setting the cell's value.
newValue	Required **Long**. The new value for the cell.

Remarks

Setting the **ResultFromInt** property is similar to setting a cell's **Result** property. The difference is that the **ResultFromInt** property accepts an integer for the value of the cell, whereas the **Result** property accepts a floating point number.

You can specify *unitsNameOrCode* as an integer or a string value. If the string is invalid, an error is generated. For example, the following statements all set *unitsNameOrCode* to inches.

Cell.ResultFromInt(**visInches**) = *newValue*

Cell.ResultFromInt(65) = *newValue*

Cell.ResultFromInt("in") = *newValue* where "in" can also be any of the alternate strings representing inches, such as "inch", "in.", or "i".

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

If the cell's formula is protected with a GUARD function, use the **ResultFromIntForce** property.

ResultFromIntForce property

Example

Sets the value of a cell to an integer value, even if the cell's formula is protected with the GUARD function.

Version added

4.0

Syntax

object.**ResultFromIntForce**(*unitsNameOrCode*) = *newVa*

object	Required. An expression that returns a **Cell** object.
unitsNameOrCode	Required **Variant**. The units to use when setting the cell's valu
newValue	Required **Long**. The new value for the cell.

Remarks

Use the **ResultFromIntForce** property to set a cell's value even if the cell's formula is protected with a GUARD function. Otherwise, it is identical in behavior to the **ResultFromInt** property.

ResultInt property

Example

Gets the value of a cell expressed as an integer.

Version added

4.0

Syntax

intRet = object.**ResultInt**(*unitsNameOrCode*,*roundFlag*)

intRet	**Long**. The cell's value returned as an integer.
object	Required. An expression that returns a **Cell** object.
unitsNameOrCode	Required **Variant**. The units to use when retrieving the cell's value.
roundFlag	Required **Integer**. Zero (0) to truncate the value; non-zero to round it.

Remarks

Setting the **ResultInt** property is similar to a setting a cell's **Result** property. The

difference is that the **ResultInt** property returns an integer for the value of the cell, whereas the **Result** property returns a floating point number.

You can specify *unitsNameOrCode* as an integer or a string value. If the string is invalid, an error is generated. For example, the following statements all set *unitsNameOrCode* to inches.

intRet = **Cell.ResultInt**(**visInches,** *roundFlag*)

intRet = **Cell.ResultInt**(65, *roundFlag*)

intRet = **Cell.ResultInt**("in", *roundFlag*) where "in" can also be any of the alternate strings representing inches, such as "inch", "in.", or "i".

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

The following constants for *roundFlag* are declared in the Visio type library in member **VisRoundFlags**.

Constant	Value	Description
visTruncate	0	Truncate the result
visRound	1	Round the result

ResultIU property

Example

Gets or sets a cell's value in internal units.

Version added

2.0

Syntax

retVal = *object*.**ResultIU**

object.**ResultIU** = *newValue*

retVal	**Double**. The cell's value in internal units.
object	Required. An expression that returns a **Cell** object.
newValue	Required **Double**. The new value for the cell.

Remarks

Use the **ResultIU** property to set the value of an unguarded cell. If a cell's formula is protected with a GUARD function, the formula is not changed and an error is generated.

The units default to the Visio internal units, which are inches for distance and radians for angles.

ResultIUForce property

Example

Sets a cell's value in internal units, even if the cell's formula is protected with the GUARD function.

Version added

2.0

Syntax

object.**ResultIUForce** = *newValue*

object	Required. An expression that returns a **Cell** object.
newValue	Required **Double**. The new value for the cell.

Remarks

The cell's units default to the Visio internal units, which are inches for distance and radians for angles.

ResultStr property

Gets the value of a cell expressed as a string.

Version added

4.0

Syntax

stringRet = *object*.**ResultStr**(*unitsNameOrCode*)

stringRet	**String**. The cell's value returned as a string.
object	Required. An expression that returns a **Cell** object.
unitsNameOrCode	Required **Variant**. The units to use when retrieving the value.

Remarks

Setting the **ResultStr** property is similar to setting a cell's **Result** property. The difference is that **ResultStr** property returns a string for the value of the cell, whereas the **Result** property returns a floating point number.

You can specify *unitsNameOrCode* as an integer or a string value. If the string is invalid, an error is generated. For example, the following statements all set *unitsNameOrCode* to inches.

stringRet = **Cell.ResultStr**(**visInches**)

stringRet = **Cell.ResultStr**(65)

stringRet = **Cell.ResultStr**("in") where "in" can also be any of the alternate strings representing inches, such as "inch", "in.", or "i".

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

Passing a zero (0) is sufficient for getting the value of text string cells.

You can use the **ResultStr** property to convert between units. For example, you can get the value in inches, then get an equivalent value in centimeters.

The **ResultStr** property is useful for filling controls such as edit boxes with the value of a cell.

RightMargin property

Example

Specifies the right margin, which is used when printing.

Version added

4.0

Syntax

retVal = object.**RightMargin**([*unitsNameOrCode*])

object.**RightMargin**([*unitsNameOrCode*]) = *newValue*

retVal	**Double**. The margin value expressed in the given units.
object	Required. An expression that returns a **Document** object.
unitsNameOrCode	Optional **Variant**. The units to use when retrieving or setting the margin value.
newValue	Required **Double**. The new margin value.

Remarks

If *unitsNameOrCode* is not provided, the **RightMargin** property will default to

internal drawing units (inches).

The **RightMargin** property corresponds to the **Right** setting in the **Print Setup** dialog box (click **Page Setup** on the **File** menu, and then click **Setup**).

You can specify *unitsNameOrCode* as an integer or a string value. If the string is invalid, an error is generated. For example, the following statements all set *unitsNameOrCode* to inches.

Cell.RightMargin(**visInches**) = *newValue*

Cell.RightMargin (65) = *newValue*

Cell.RightMargin ("in") = *newValue* where "in" can also be any of the alternate strings representing inches, such as "inch", "in.", or "i".

For a complete list of valid unit strings along with corresponding Automation constants (integer values), see About units of measure.

Automation constants for representing units are declared by the Visio type library in member **VisUnitCodes**.

RootShape property

See also Example

Returns the top-level shape of an instance if this shape is part of a master instance.

Version added

2002

Syntax

objRet = *object*.**RootShape**

objRet	The **Shape** object that is the top level shape of this instance.
object	Required. An expression that returns a **Shape** object.

Remarks

If this shape is not part of a master instance the **RootShape** property returns **Nothing**.

Row property

Example

Returns the row index of a cell or layer.

Version added

4.0

Syntax

intRet = *object*.**Row**

intRet	**Integer**. The index of the row that defines the cell or layer.
object	Required. An expression that returns a **Cell**, **Hyperlink**, **Layer**, or **Section** object.

RowCount property

Returns the number of rows in a ShapeSheet section.

Version added

2.0

Syntax

retVal = *object*.**RowCount** (*section*)

retVal	**Integer**. The number of rows in the section.
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The section to count.

Remarks

The section argument must be a section constant. For a list of section constants, see the **AddSection** method.

Use the **RowCount** property primarily with sections that contain a variable number of rows, such as Geometry and Connection Points sections. The value

returned by the **RowCount** property for sections that have a fixed number of rows is the number of rows in the section that possess at least one cell whose value is local to the shape. This is opposed to rows whose cells are all inherited from a master or style, which is typically better because Visio doesn't need to store as much information. In the ShapeSheet window, cells with local values appear in blue, and cells with inherited values appear in black. Using Automation, you can determine if a cell is inherited using the **IsInherited** property.

RowExists property

Example

Determines whether a ShapeSheet row exists.

Version added

4.0

Syntax

intRet = *object*.**RowExists**(*section*, *row*, *fExistsLocally*)

intRet	**Integer. False** (0) if row doesn't exist; otherwise, **True** (-1).
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The row's section index.
row	Required **Integer**. The row's row index.
fExistsLocally	Required **Integer**. The scope of the search.

Remarks

If *fExistsLocally* is **False** (0), the **RowExists** property returns **True** if the object either contains or inherits the specified row.

If *fExistsLocally* is **True** (non-zero), the **RowExists** property returns **True** only if the object contains the specified row locally; if the row is inherited, the **RowExists** property returns **False**.

For a list of row index values, see the **AddRow** method or view the Visio type library for the members of class **visRowIndices**. For a list of section index values, see the **AddSection** method or view the Visio type library for the members of class **visSectionIndices**.

RowIndex property

Example

Gets or sets the docking order of a **MenuSet** or **Toolbar** object in relation to other items in the same docking area.

Version added

2000

Syntax

intLong = *object*.**RowIndex**

object. **RowIndex** = *intLong*

object	Required. An expression that returns a **MenuSet** or **Toolbar** object.
intLong	Required **Integer**.

Remarks

Objects with lower numbers are docked first. Several items can share the same row index. If two or more items share the same row index, the item most

recently assigned is displayed first in its group.

Constants representing the first and last positions (see the following table) are declared by the Visio type library in member **visUIBarRow**.

Constant	Value
visBarRowFirst	0
visBarRowLast	-1

RowName[U] property

Example

Gets or sets the name of the row that contains the **Cell** object.

Version added

4.0

Syntax

strRet = *object*.**RowName**

object.**RowName** = *stringExpression*

strRet	**String**. The current name of the row.
object	Required. An expression that returns a **Cell** object.
stringExpression	Required **String**. The new name to assign to the row.

Remarks

If the cell is in a row in a shape's User-defined Cells, Custom Properties, or Connection Points section, the **RowName** property can get or set the name of the row. If the cell is not in one of these sections, attempting to get or set the name

generates an error.

The Connection Points section can contain either named or unnamed rows, but not a combination of the two. Getting the name of an unnamed Connection Points row returns a zero-length string ("") and does not generate an error. Setting the name of an unnamed row in a Connection Points row assigns the name to the target row and converts all remaining rows in the section to named rows, using their default names (Row_1, Row_2, and so on). Assigning a zero-length string ("") to a named row in a Connection Points section resets the named row to its default name, but has no effect on an unnamed Connection Points row.

When you change a row name, any cell objects referring to cells in that row become invalid and you must reassign them. Also, if other Connection Points rows become named as a result of a row name change, you must also reassign references to cells in those rows.

Note Beginning with Visio 2000, you can refer to Visio shapes, masters, styles, pages, rows, and layers using local and universal names. When a user names a shape, for example, the user is specifying a local name. Universal names are not visible through the user interface. As a developer, you can use universal names in a program when you don't want to change a name each time a solution is localized. Use the **RowName** property to get or set an object's local row name. Use the **RowNameU** property to get or set an object's universal row name.

RowsCellCount property

Example

Returns the number of cells in a row of a ShapeSheet section.

Version added

2.0

Syntax

intRet = *object*.**RowsCellCount** (*section*, *row*)

intRet	**Integer**. The number of cells in the row.
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The index of the section that contains the row.
row	Required **Integer**. The index of the row to count.

Remarks

Use section and row index constants declared by the Visio type library in members **VisSectionIndices** and **VisRowIndices**. They are also listed in the

AddRow and **AddSection** method topics.

RowType property

Gets or sets the type of a row in a Geometry, Connection Points, Controls, or Tabs ShapeSheet section.

Version added

2.0

Syntax

retVal = *object*.**RowType** (*section*, *row*)

object.**RowType** (*section*, *row*) = *rowTag*

retVal	**Integer**. The current type of the row.
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The index of the section that contains the row.
row	Required **Integer**. The index of the row.
rowTag	Required **Integer**. The new type for the row.

Remarks

After you change a row's type, the new row type may or may not have the same cells. Your program must provide the appropriate formulas for the new or changed cells.

The *rowType* argument specifies the type of row you want. You can use any of the following constants declared by the Visio type library in member **VisRowTags**.

Constant	Value
visTagComponent	137
visTagMoveTo	138
visTagLineTo	139
visTagArcTo	140
visTagInfiniteLine	141
visTagEllipse	143
visTagEllipticalArcTo	144
visTagSplineBeg	165
visTagSplineSpan	166
visTagPolylineTo	193
visTagNURBSTo	195
visTagTab0	136
visTagTab2	150
visTagTab10	151
visTagTab60	181
visTagCnnctPt	153
visTagCnnctNamed	185
visTagCtlPt	162
visTagCtlPtTip	170

If an inappropriate row tag is passed or the row does not exist, no changes occur and an error is returned.

Use the **RowName** property to transition from unnamed to named Connection Points rows.

See the **AddRow** method for a list of valid row constants and row tag constants.

See the **AddSection** method for a list of valid section constants.

RunBegin property

Example

Returns the beginning index of a type of run—a sequence of characters that share a particular attribute, such as character, paragraph, or tab formatting; or a word, paragraph, or field.

Version added

3.0

Syntax

intRet = *object*.**RunBegin**(*runType*)

intRet	**Long**. The beginning index of the run.
object	Required. An expression that returns a **Characters** object.
runType	Required **Integer**. The type of run to get.

Remarks

In a ShapeSheet window, each row in the Character and Paragraph sections represents a run of the corresponding format in a shape's text. Certain words may be bold or italic, or one paragraph may be centered and another left-aligned.

Each change of format represents a run of that format. Similarly, delimiters such as spaces and paragraph marks represent the beginning and end of words, paragraphs, and fields.

In addition, you can retrieve rows that represent runs of character, paragraph, and tab formats by specifying a row index as an argument to the **CellsSRC** property of a shape.

Use the **RunBegin** property to determine the beginning of a sequence of identically formatted characters or the beginning of a word, paragraph, or field. You can check the **IsField** property to determine whether a run is a field.

The index that the **RunBegin** property returns is less than or equal to the beginning index of a **Characters** object. If the **Begin** property of the **Characters** object is already at the start of a run, the value of the **RunBegin** property is equal to the value of **Begin**.

Use the *runType* argument to specify the type of run you want. You can also use any of the following constants declared by the Visio type library in member **VisRunTypes**.

Constant	Value	Description
visCharPropRow	1	Reports runs of characters with common character properties. Corresponds to set of characters covered by one row in shape's Character section.
visParaPropRow	2	Reports runs of characters with common paragraph properties. Corresponds to set of characters covered by one row in shape's Paragraph section.
visTabPropRow	3	Reports runs of characters with common tab properties. Corresponds to set of characters covered by one row in shape's Tabs section.
visWordRun	10	Reports runs whose boundaries are between successive words in

		shape's text. Mimics double-clicking to select text.
visParaRun	11	Reports runs whose boundaries are between successive paragraphs in shape's text. Mimics triple-clicking to select text.
visFieldRun	20	Reports runs whose boundaries are between characters that are and aren't the result of the expansion of a text field, or between characters that are the result of the expansion of distinct text fields.

RunEnd property

Example

Returns the ending index of a type of run—a sequence of characters that share a particular attribute, such as character, paragraph, or tab formatting; or a word, paragraph, or field.

Version added

3.0

Syntax

intRet = *object*.**RunEnd**(*runType*)

intRet	**Long**. The ending index of the run.
object	Required. An expression that returns a **Characters** object.
runType	Required **Integer**. The type of run to get.

Remarks

In a ShapeSheet window, each row in the Character and Paragraph sections represents a run of the corresponding format in a shape's text. Certain words may be bold or italic, or one paragraph may be centered and another left-aligned.

Each change of format represents a run of that format. Similarly, delimiters such as spaces and paragraph marks represent the beginning and end of words, paragraphs, and fields.

In addition, you can retrieve rows that represent runs of character, paragraph, and tab formats by specifying a row index as an argument to the **CellsSRC** property of a shape.

Use the **RunEnd** property to determine the end of a sequence of identically formatted characters or the end of a word, paragraph, or field. You can check the **IsField** property to determine whether a run is a field.

The index that the **RunEnd** property returns is greater than or equal to the ending index of a **Characters** object. If the **End** property of the **Characters** object is already at the end of a run, the value of the **RunEnd** property is equal to the value of the **End** property.

Use the *runType* argument to specify the type of run you want. You can also use any of the constants declared by the Visio type library in **VisRunTypes**. To find a list of *runType* values, see the **RunBegin** property.

Saved property

See also

Determines whether a document has any unsaved changes.

Version added

2.0

Syntax

boolRet = *object*.**Saved**

object.**Saved** = *boolExpression*

boolRet	**Boolean**. **True** if the document has no unsaved changes; otherwise, **False**.
object	Required. An expression that returns a **Document** object.
boolExpression	Required **Boolean**. **True** to indicate the document is saved; **False** to indicate unsaved changes.

Remarks

Setting the **Saved** property for a document to **True** should be done with caution.

If you set the **Saved** property to **True** and a user, or another program, makes changes to the document before it is closed those changes will be lost—Visio does not provide a prompt to save the document.

A document that contains embedded or linked OLE objects may report itself as unsaved even if the document's **Saved** property is set to **True**.

SavePreviewMode property

See also Example

Determines how a preview picture is saved in a file.

Version added

4.0

Syntax

retVal = *object*.**SavePreviewMode**

object.**SavePreviewMode** = *newVal*

retVal	**VisSavePreviewMode**. The type of preview picture that Visio saves.
object	Required. An expression that returns a **Document** object.
newVal	Required **VisSavePreviewMode**. The type of preview picture that Visio saves.

Remarks

The value of the **SavePreviewMode** property is equivalent to the **Preview** setting on the **Summary** tab in the **Properties** dialog box (click **Properties** on the **File** menu). A preview of the first page appears in the **Open** dialog box. The value of *retVal* and *newVal* can be one of the following **VisSavePreviewMode** constants.

Constant	Value	Description

visSavePreviewNone	0	No preview picture.
visSavePreviewDraft1st	1	The first page with only Visio shapes. Does not include embedded objects, text, or gradient fills.
visSavePreviewDetailed1st	2	The first page with all objects.
visSavePreviewDraftAll	4	All file pages with only Visio shapes. Does not include embedded objects, text, or gradient fills.
visSavePreviewDetailedAll	8	All file pages with all objects.

ScreenUpdating property

Determines whether the screen is updated (redrawn) during a series of actions.

Version added

3.0

Syntax

intRet = *object*.**ScreenUpdating**

object.**ScreenUpdating** = *intExpression*

intRet	**Integer**. Zero (0) if screen updating is off; -1 if screen updating is on.
object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. Zero (0) to turn screen updating off; non-zero to turn screen updating on.

Remarks

Use the **ScreenUpdating** property to increase performance during a series of

actions. For example, you can turn off screen updating while a series of shapes are created so the screen is not redrawn after each shape appears, and then turn screen updating on to update the screen.

If you send a large number of commands to a Visio instance while screen updating is turned off, the Visio instance may redisplay the screen occasionally in order to flush its buffers.

If a program neglects to turn screen updating on after turning it off, the Visio instance turns screen updating back on when a user performs an operation.

Note Beginning with Visio 2000, the **ShowChanges** property is included. The **ShowChanges** and **ScreenUpdating** properties are similar in that they are both designed to increase performance during a series of actions, but they work differently. Setting the **ShowChanges** property also sets the **ScreenUpdating** property, but setting the **ScreenUpdating** property does not set the **ShowChanges** property. For information comparing these two properties, see the **ShowChanges** property.

Section property

Example

Returns the requested **Section** object belonging to a shape or style.

Version added

4.0

Syntax

objRet = *object*.**Section**(*index*)

objRet	The **Section** object that corresponds to the index.
object	Required. An expression that returns a **Shape** or **Style** object.
index	Required **Integer**. A section index.

Remarks

Constants representing sections are prefixed with **visSection** and are declared by the Visio type library in **VisSectionIndices**. You can also view a list of constants in the **AddSection** method.

Section property (Cell object)

Example

Returns the index of the cell's section.

Version added

4.0

Syntax

intRet = object.**Section**

intRet	**Integer**. The index of the cell's section.
object	Required. An expression that returns a **Cell** object.

Remarks

Constants representing sections are prefixed with **visSection** and are declared by the Visio type library in **VisSectionIndices**. You can also view a list of constants in the **AddSection** method.

SectionExists property

Example

Determines whether a ShapeSheet section exists for a particular shape.

Version added

4.0

Syntax

intRet = *object*.**SectionExists**(*section*, *fExistsLocally*)

intRet	**Integer**. **False** (0) if section doesn't exist; otherwise, **True** (1).
object	Required. An expression that returns a **Shape** object.
section	Required **Integer**. The section index.
fExistsLocally	Required **Integer**. The scope of the search.

Remarks

If *fExistsLocally* is **False** (0), the **SectionExists** property returns **True** if the object either contains or inherits the section. If *fExistsLocally* is **True** (non-zero), the **SectionExists** property returns **True** only if the object contains the section

locally; if the section is inherited, the **SectionExists** property returns **False**.

Constants representing sections are prefixed with **visSection** and are declared by the Visio type library in **VisSectionIndices**. You can also view a list of constants in the **AddSection** method.

Selection property

Returns a **Selection** object that represents what is presently selected in the window.

Version added

2.0

Syntax

objRet = *object*.**Selection**

objRet	A **Selection** object.
object	Required. An expression that returns a **Window** object.

Remarks

The **Selection** object is independent of the selection in the window, which can subsequently change as a result of user actions.

A **Selection** object is a set of shapes in a common context on which you can perform actions. A **Selection** object is analogous to more than selected shapes in

a drawing window. Once you retrieve a **Selection** object, you can change the set of shapes the object represents using the **Select** method.

SetID property

Example

Returns the set ID of an **AccelTable**, **Menuset**, or **Toolbar** object in its collection.

Version added

4.0

Syntax

intRet = *object*.**SetID**

intRet	**Long**. The set ID of the object.
object	Required. An expression that returns an **AccelTable**, **MenuSet**, or **ToolbarSet** object.

Remarks

Each **AccelTable**, **MenuSet**, and **ToolbarSet** object has a set ID that corresponds to a Visio window context. For **MenuSet** objects, IDs also correspond to shortcut menu sets. And for **ToolbarSet** objects, they also correspond to drop-down menus under toolbar buttons (such as **Fill Color** or

Line Weight).

You can use the set ID to retrieve an object from its collection with the **ItemAtID** property. You can also set the set ID of an object using the **AddAtID** method.

Valid set ID values are declared by the Visio type library in **VisUIObjSets**.

- ID constants for AccelTable objects
- ID constants for MenuSet objects
- ID constants for ToolbarSet objects

Shape property

Returns the **Shape** object that owns a **Cell** or **Characters** object or that is associated with an **OLEObject** or **Hyperlink** object.

Version added

3.0

Syntax

objRet = *object*.**Shape**

objRet	The **Shape** object that contains or is associated with the object.
object	Required. An expression that returns an object in the **Applies to** list.

ShapeHelp property

See also Example

Gets or sets the help string used when the user clicks **Help** on the shortcut menu of a master shortcut.

Version added

2000

Syntax

strRet = *object*.**ShapeHelp**

object.**ShapeHelp** = *strExpression*

strRet	**String**. The current help string.
object	Required. An expression that returns a **MasterShortcut** object.
strExpression	Required **String**. The new help string.

Remarks

If the help string is blank, the **Help** command uses the help string defined by the shortcut's target master, determined by the **Help** property of that master's top-level shape.

Shapes property

Returns the **Shapes** collection for a page, master, or group.

Version added

2.0

Syntax

objsRet = *object*.**Shapes**

objsRet	The **Shapes** collection of the object.
object	Required. An expression that returns a **Page**, **Master**, or **Shape** object that owns the collection.

Shift property

Example

Determines whether the SHIFT key is a modifier for an **AccelItem** object.

Version added

4.0

Syntax

intRet = *object*.**Shift**

object.**Shift** = *intExpression*

intRet	**Integer**. **True** (-1) if modified by SHIFT key; otherwise, **False** (0).
object	Required. An expression that returns an **AccelItem** object.
intExpression	Required **Integer**. **True** (non-zero) if modified by SHIFT key; otherwise, **False** (0).

ShowChanges property

Example

Determines whether the screen is updated (redrawn) during a series of actions.

Version added

2000

Syntax

boolRet = *object*.**ShowChanges**

object.**ShowChanges** = *boolExpression*

boolRet	**Boolean**. **True** if the screen is updated for each document change; **False** if it is not.
object	Required. An expression that returns an **Application** object.
boolExpression	Required **Boolean**. **True** to update the screen for each document change; **False** to leave the screen unchanged.

Remarks

Use the **ShowChanges** property to increase performance during a series of

actions. For example, you can set the **ShowChanges** property to **False** while a series of shapes are created so the screen is not redrawn after each shape appears, and then set it to **True** to update the screen.

If a program neglects to turn the **ShowChanges** property on after turning it off, the Visio instance will turn it back on when the user performs an operation.

The **ShowChanges** property is similar to the **ScreenUpdating** property, which was implemented in Visio 3.0. In most cases using the **ShowChanges** property is preferable to using the **ScreenUpdating** property. Setting the **ShowChanges** property automatically sets the **ScreenUpdating** property; however, setting the **ScreenUpdating** property does not set the **ShowChanges** property.

When **ShowChanges** is **False**, the Visio instance will not refresh the screen as documents change. All shapes in drawing and stencil windows are deselected and the Visio instance won't allow programs to change the selections of windows.

When only **ScreenUpdating** is **False**, the Visio instance will occasionally refresh the screen as documents change. **ScreenUpdating** does not cause deselects to occur or restrict selection changes.

The Visio instance will usually run faster when both the **ShowChanges** and **ScreenUpdating** properties are **False** than when only the **ScreenUpdating** property is **False**. When both the **ShowChanges** and **ScreenUpdating** properties are **False**, the Visio views will not react to document changes until the **ShowChanges** property becomes **True**. This can cause noticeable delays after a program has completed a sequence of many operations. To cause some changes to occur as they happen, set **ScreenUpdating** to **True** immediately after setting **ShowChanges** to **False**. This can shorten the delay that occurs after **ShowChanges** becomes **True**, but will probably lengthen the time to complete the overall sequence of actions.

ShowConnectPoints property

Example

Determines whether connection points are shown in a window.

Version added

4.5

Syntax

intRet = *object*.**ShowConnectPoints**

object.**ShowConnectPoints** = *intExpression*

intRet	**Integer**. **True** (-1) if connection points are shown; otherwise, **False** (0).
object	Required. An expression that returns a **Window** object.
intExpression	Required **Integer**. **True** (-1) to show connection points; **False** (0) to hide connection points.

Remarks

Using the **ShowConnectPoints** property is equivalent to clicking **Connection Points** on the **View** menu.

ShowGrid property

Determines whether a grid is shown in a window.

Version added

4.5

Syntax

intRet = *object*.**ShowGrid**

object.**ShowGrid** = *intExpression*

intRet	**Integer**. **True** (-1) if grids are showing; **False** (0) if grids are hidden.
object	Required. An expression that returns a **Window** object.
intExpression	Required **Integer**. **True** (non-zero) to show a grid; **False** (0) to hide a grid.

Remarks

Setting the **ShowGrid** property is equivalent to clicking **Grid** on the **View**

menu.

ShowGuides property

Example

Determines whether guides are shown in a window.

Version added

4.5

Syntax

intRet = *object*.**ShowGuides**

object.**ShowGuides** = *intExpression*

intRet	**Integer**. **True** (-1) if guides are showing; **False** (0) if guides are hidden.
object	Required. An expression that returns a **Window** object.
intExpression	Required **Integer**. **True** (non-zero) to show guides; **False** (0) to hide guides.

Remarks

Setting the **ShowGuides** property is equivalent to clicking **Guides** on the **View**

menu.

ShowMenus property

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property determined whether menus were shown (see the ShowToolbar property).

ShowPageBreaks property

Example

Determines whether page breaks are shown in a window.

Version added

4.5

Syntax

intRet = *object*.**ShowPageBreaks**

object.**ShowPageBreaks** = *intExpression*

intRet	**Integer**. **True** (-1) if page breaks are showing; **False** (0) if page breaks are hidden.
object	Required. An expression that returns a **Window** object.
intExpression	Required **Integer**. **True** (non-zero) to show page breaks; **False** (0) to hide page breaks.

Remarks

Setting the **ShowPageBreaks** property is equivalent to clicking **Page Breaks** on

the **View** menu.

ShowPageTabs property

See also Example

Determines whether page tab controls are shown in the drawing window.

Version added

2002

Syntax

boolRet = *object*.**ShowPageTabs**

object.**ShowPageTabs** = *boolExpression*

boolRet	**Boolean**. **True** if page tabs are showing; otherwise, **False**.
object	Required. An expression that returns a **Window** object.
boolExpression	Required **Boolean**. **True** to show page tabs; otherwise, **False**.

ShowProgress property

Determines whether a progress indicator is shown while performing certain operations.

Version added

4.1

Syntax

intRet = *object*.**ShowProgress**

object.**ShowProgress** = *intExpression*

intRet	**Integer**. **True** (-1) if a progress indicator is showing; otherwise, **False** (0).
object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. **True** (non-zero) to show a progress indicator; otherwise, **False** (0).

Remarks

If you want to perform an operation, such as printing, that typically displays a progress indicator but you don't want the progress indicator to appear, set the **ShowProgress** property to **False** (0). By default, the **ShowProgress** property is **True** (non-zero).

In most cases you should restore the setting to its prior value when you've completed the operation.

ShowRulers property

Determines whether rulers are shown in the drawing window.

Version added

4.5

Syntax

intRet = *object*.**ShowRulers**

object.**ShowRulers** = *intExpression*

intRet	**Integer**. **True** (-1) if rulers are showing; **False** (0) if rulers are hidden.
object	Required. An expression that returns a **Window** object.
intExpression	Required **Integer**. **True** (non-zero) to show rulers; **False** (0) to hide rulers.

Remarks

Setting the **ShowRulers** property is the same as clicking **Rulers** on the **View**

menu.

ShowScrollBars property

Example

Determines whether scroll bars are shown in the drawing window.

Version added

2002

Syntax

intRet = *object*.**ShowScrollBars**

object.**ShowScrollBars** = *intExpression*

intRet	**Integer**. Non-zero if one or more scroll bars are showing; otherwise, zero (0).
object	Required. An expression that returns a **Window** object.
intExpression	Required **Integer**. See Remarks for possible values.

Remarks

The *intExpression* argument can be any combination of the following **VisScrollbarStates** constants, which are declared in the Visio type library.

Constant	Value
visScrollBarNeither	&H0
visScrollBarHoriz	&H1
visScrollBarVert	&H4
visScrollBarBoth	&H5

ShowStatusBar property

Determines whether a status bar is shown.

Version added

4.5

Syntax

intRet = *object*.**ShowStatusBar**

object.**ShowStatusBar** = *intExpression*

intRet	**Integer**. **True** (-1) if the status bar is showing; otherwise, **False** (0).
object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. **True** (-1) to show a status bar; **False** (0) to hide a status bar.

Remarks

The **ShowStatusBar** property persists each time you run the application. The

ShowMenus and **ShowToolbar** properties are valid for a Visio instance only.

ShowToolbar property

Example

Determines whether toolbars and menu bars are visible.

Version added

5.0

Syntax

intRet = *object*.**ShowToolbar**

object.**ShowToolbar** = *intExpression*

intRet	**Integer**. **True** (-1) if toolbars and menu bars are showing; **False** (0) if toolbars and menu bars are hidden.
object	Required. An expression that returns an **Application** object.
intExpression	Required **Integer**. **True** (non-zero) to show toolbars and menu bars; **False** (0) to hide toolbars and menu bars.

Remarks

The **ShowToolbar** property is valid for a Visio instance only. The

ShowStatusBar property persists each time you run the application.

SnapAngles property

Example

Determines the degree of the angle that is drawn when isometric angle lines is chosen as a shape extension option.

Version added

2002

Syntax

retVal = *object*.**SnapAngles**

object.**SnapAngles** = *newVal*

retVal	**Double**. An array of up to 10 entries containing the current degree(s) of the isometric angle shape extensions. Separate multiple values with commas.
object	Required. An expression that returns a **Document** object.
newVal	Required **Double**. An array of up to 10 entries containing new degree values(s) of the isometric angle shape extensions. Separate multiple values with commas.

Remarks

The value of the **SnapAngles** property is equivalent to the value of the **Isometric angles (degs)** field on the **Advanced** tab in the **Snap & Glue** dialog box (on the **Tools** menu, click **Snap & Glue**).

SnapEnabled property

See also Example

Determines whether snap is active in the document.

Version added

2002

Syntax

retVal = *object*.**SnapEnabled**

object.**SnapEnabled** = *newVal*

retVal	**Boolean**. **True** if snap is active; otherwise, **False.**
object	Required. An expression that returns a **Document** object.
newVal	Required **Boolean**. **True** to enable snap behavior; **False** to disable snap behavior.

Remarks

The value of the **SnapEnabled** property is equivalent to selecting the **Snap** check box on the **General** tab in the **Snap & Glue** dialog box (on the **Tools** menu, click **Snap & Glue**).

SnapExtensions property

See also Example

Determines the shape extensions that are active in a document.

Version added

2002

Syntax

retVal = *object*.**SnapExtensions**

object.**SnapExtensions** = *newVal*

retVal	**VisSnapExtensions**. The shape extension options currently active in a document.
object	Required. An expression that returns a **Document** object.
newVal	Required **VisSnapExtensions**. The shape extension options to activate in a document.

Remarks

You can also set this value by checking options in the **Shape extension options** box on the **Advanced** tab in the **Snap & Glue** dialog box (on the **Tools** menu click **Snap & Glue**).

The **SnapExtensions** property can be any combination of the following **VisSnapExtensions** constants, which are declared in the Visio type library.

Constant	Value

visSnapExtNone	&H0
visSnapExtAlignmentBoxExtension	&H1
visSnapExtCenterAxes	&H2
visSnapExtCurveTangent	&H4
visSnapExtEndpoint	&H8
visSnapExtMidpoint	&H10
visSnapExtLinearExtension	&H20
visSnapExtCurveExtension	&H40
visSnapExtEndpointPerpendicular	&H80
visSnapExtMidpointPerpendicular	&H100
visSnapExtEndpointHorizontal	&H200
visSnapExtEndpointVertical	&H400
visSnapExtEllipseCenter	&H800
visSnapExtIsometricAngles	&H1000

SnapSettings property

See also Example

Determines the objects that shapes snap to when snap is active in the document.

Version added

2002

Syntax

retVal = *object*.**SnapSettings**

object.**SnapSettings** = *newVal*

retVal	**VisSnapSettings**. The objects in a document that shapes snap to.
object	Required. An expression that returns a **Document** object.
newVal	Required **VisSnapsettings**. The objects in a document that shapes snap to.

Remarks

The value of the **SnapSettings** property is equivalent to selecting check boxes under **Snap to** on the **General** tab in the **Snap & Glue** dialog box (on the **Tools** menu click **Snap & Glue**).

The **SnapSettings** property can be any combination of the following **VisSnapSettings** constants, which are declared in the Visio type library.

Constant	Value	Description

visSnapToNone	&H0	Snap to nothing.
visSnapToRulerSubdivisions	&H1	Snap to tick marks on the ruler.
visSnapToGrid	&H2	Snap to the grid.
visSnapToGuides	&H4	Snap to guides.
visSnapToHandles	&H8	Snap to selection handles.
visSnapToVertices	&H10	Snap to vertices.
visSnapToConnectionPoints	&H20	Snap to connection points.
visSnapToGeometry	&H100	Snap to the visible edges of shapes.
visSnapToAlignmentBox	&H200	Snap to the alignment box.
visSnapToExtensions	&H400	Snap to shape extensions options.
visSnapToDisabled	&H8000	Disable snap.
visSnapToIntersections	&H10000	Snap to intersections.

SolutionXMLElement property

Example

Contains solution-specific, well-formed XML data stored with a document.

Version added

2002

Syntax

xmlData = *object*.**SolutionXMLElement** (*elementName*)

object.**SolutionXMLElement** (*elementName*) = *xmlValue*

xmlData	**String**. The current XML data stored in *elementName*.
object	Required. An expression that returns a **Document** object.
elementName	Required **String**. The case-sensitive name of the SolutionXML data element.
xmlValue	Required **String**. The new valid, well-formed XML data to store in *elementName*.

Remarks

The value of *elementName* must match the value of the SolutionXML element's Name attribute. For example, if a solution's XML data began with the statement <SolutionXML Name='somename'>, use the *elementName* "somename" to retrieve that data.

If *elementName* already exists, the **SolutionXMLElement** property overwrites existing XML data. Use the **SolutionXMLElementExists** property before writing XML data to avoid losing data unintentionally.

If *elementName* does not exist, the **SolutionXMLElement** property creates an element by that name.

Because your XML data is validated when you write it, you will typically perform this operation during a document save event for performance reasons.

SolutionXMLElementCount property

Example

Returns the number of SolutionXML elements in a document.

Version added

2002

Syntax

intRet = *object*.**SolutionXMLElementCount**

intRet	**Long**. The number of SolutionXML elements in the document.
object	Required. An expression that returns a **Document** object.

Remarks

The first element in the document has an index of 1.

SolutionXMLElementExists property

Example

Indicates whether a named SolutionXML element exists in the document.

Version added

2002

Syntax

boolRet = object.**SolutionXMLElementExists** (*elementN*

boolRet	**Boolean**. **True** (-1) if a SolutionXML element named *elementN* exists; **False** (0) if it does not.
object	Required. An expression that returns a **Document** object.
elementName	Required **String**. The case-sensitive name of the SolutionXML element.

Remarks

Because the **SolutionXMLElement** property can overwrite existing XML data, always use the **SolutionXMLElementExists** property to verify whether *elementName* already exists in the document.

SolutionXMLElementName property

Example

Returns the name of the SolutionXML element.

Version added

2002

Syntax

strRet = *object*.**SolutionXMLElementName**(*index*)

strRet	**String.** The case-sensitive name of the SolutionXML element.
object	Required. An expression that returns a **Document** object.
index	Required **Long**. The index of the SolutionXML element in the document.

Remarks

The only way to retrieve SolutionXML data is by name. You can use the **SolutionXMLElementName** property to get the element name to pass to the **SolutionXMLElement** property.

Spacing property

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property determined the spacing between menus, menu items, and toolbar items.

SpatialNeighbors property

Returns a **Selection** object that represents the shapes that meet certain criteria in relation to a specified shape.

Version added

2000

Syntax

objRet = *object*.**SpatialNeighbors**(*relation*, *tolerance*, *fla*

objRet	A **Selection** object.
object	Required. An expression that returns a **Shape** object.
relation	Required **Integer**. An integer describing the type of relation
tolerance	Required **Double**. A distance in internal drawing units with coordinate space defined by the parent shape.
flags	Required **Integer**. Flags that influence the type of entries ret
resultRoot	Optional **Variant**. A **Shape** object that represents a page or

Remarks

For values of the *relation* argument, see the SpatialRelation property.

The *flags* argument can be any combination of the values of the constants defined in the following table. These constants are also defined in **VisSpatialRelationFlags** in the Visio type library.

Constant	Value	Description
visSpatialIncludeGuides	&H2	Consider a guide's Geometry section. By default, guides do not influence the result.
visSpatialFrontToBack	&H4	Order items front to back.
visSpatialBackToFront	&H8	Order items back to front.
visSpatialIncludeHidden	&H10	Consider hidden Geometry sections. By default, hidden Geometry sections do not influence the result.
visSpatialIgnoreVisible	&H20	Do not consider visible Geometry sections. By default, visible Geometry sections influence the result.

Use the NoShow cell to determine whether a Geometry section is hidden or visible. Hidden Geometry sections have a value of TRUE and visible Geometry sections have a value of FALSE in the NoShow cell.

Beginning with Visio 2002, if *flags* contains **VisSpatialFrontToBack**, items in the **Selection** object returned by the **SpatialNeighbors** property are ordered front to back. If **visSpatialBackToFront** is set, the items returned are ordered back to front. If this flag is not set, or if you are running an earlier version of Visio, the order is unpredictable. You can determine the order using the **Index** property of the shapes identified in the **Selection** object.

If you don't specify *resultRoot*, this property returns a **Selection** object that represents the shapes that meet certain criteria in relation to the specified shape. If you specify *resultRoot*, this property returns a **Selection** object that represents all the shapes in the **Shape** object specified by *resultRoot* that meet certain criteria in relation to the specified shape. For example, specify *resultRoot* to find all shapes within a group that are near a specified shape.

If *resultRoot* is specified but isn't on the same page or in the same master as the **Shape** object to which you are comparing it, the **SpatialNeighbors** property raises an exception and returns **Nothing**.

If *relation* is not specified, the **SpatialNeighbors** property uses all the possible relationships as criteria.

The **SpatialNeighbors** property does not consider the width of a shape's line, shadows, line ends, control points, or connection points when comparing two shapes.

SpatialRelation property

Returns an integer that represents the spatial relationship of one shape to another shape. Both shapes must be on the same page or in the same master.

Version added

2000

Syntax

relation = *object*.**SpatialRelation**(*otherShape*, *tolerance*,

relation	**Integer**. The relationship between two shapes. See Remarks fo values of this argument.
object	Required. An expression that returns a **Shape** object.
otherShape	Required. The other **Shape** object involved in the comparison.
tolerance	Required **Double**. A distance in internal drawing units with res the coordinate space defined by the **Shape** object's parent.
flags	Required **Integer**. Flags that influence the result. See Remarks values of this argument.

Remarks

- Values for *relation*
- Values for *flags*

Note The **SpatialRelation** property does not consider the width of a shape's line, shadows, line ends, control points, or connection points when comparing two shapes.

SpatialSearch property

Example

Returns a **Selection** object whose shapes meet certain criteria in relation to a point that is expressed in the coordinate space of a page, master, or group.

Version added

2000

Syntax

objRet = *object*.**SpatialSearch**(*x*, *y*, *relation*, *tolerance*, *fl*

objRet	A **Selection** object.
object	Required. An expression that returns a **Page**, **Master**, or **Shape** object.
x	Required **Double**; *x*-coordinate.
y	Required **Double**, *y*-coordinate.
relation	Required **Integer**. Any combination of the values of the consta **visSpatialContainedIn** and **visSpatialTouching**.
tolerance	Required **Double**. A distance in internal drawing units with resp to the coordinate space.

flags	Required **Integer**. Flags that influence the result.

Remarks

For values of the *relation* argument, see the SpatialRelation property.

If *relation* is not specified, the **SpatialSearch** property uses both relationships as criteria.

- Values of *flags*

The **SpatialSearch** property does not consider the width of a shape's line, shadows, line ends, control points, or connection points when comparing two shapes.

Start property

Returns the start of a **Curve** object's parameter domain.

Version added

5.0

Syntax

retVal = object.**Start**

retVal	**Double**. Starting value of a **Curve** object's parameter domain.
object	Required. An expression that returns a **Curve** object.

Remarks

The **Start** property of curve returns the coordinates of the curve's starting point. A **Curve** object describes itself in terms of its parameter domain, which is the range [Start(),End()] where Start() produces the curve's starting point.

StartupPaths property

Gets or sets the paths where Microsoft Visio looks for add-ons to run when the application is started.

Version added

Visio 4.0

Syntax

strRet = *object*.**StartupPaths**

object.**StartupPaths** = *pathsStr*

strRet	**String**. A text string containing a list of folders. Folders are separated by semicolons.
object	Required. An expression that returns an **Application** object.
pathsStr	Required **String**. A text string containing a list of folders; to indicate more than one folder, separate individual items in the path string with semicolons.

Remarks

The string passed to and received from the **StartupPaths** property is the same string shown on the **File Paths** tab in the **Options** dialog box (click **Options** on the **Tools** menu, and then click **File Paths**). This string is stored in **HKEY_CURRENT_USER\Software\Microsoft\Visio\application\StartUpPa**

When the application looks for startup files, it looks in all paths named in the **StartupPaths** property and all the subfolders of those paths. If you pass the **StartupPaths** property to the **EnumDirectories** method, it returns a complete list of fully qualified paths in which the application looks.

If a path is not fully qualified, the application looks for the folder in the folder that contains the Visio program files (*appObj*.**Path**). For example, if the Visio executable file is installed in c:\Visio, and the **StartupPaths** property is "Startup;d:\Startup", the application looks for startup files in both c:\Visio\Startup and d:\Startup.

Stat property

Returns status information for an object.

Version added

3.0

Syntax

intRet = *object*.**Stat**

intRet	**Integer**. A bit mask of status bits.
object	Required. An expression that returns an object in the **Applies to** list.

Remarks

If an object is a reference to an entity in a document, and if that document closes, the **Stat** property returns a value in which the **visStatClosed** bit is set.

If an object is a reference to an entity that has been deleted, the **Stat** property returns a value in which the **visStatDeleted** bit is set.

A Component Object Model (COM) object, such as a Visio document object, lives as long as it is held (pointed to) by a client, even if the object is logically in a deleted or closed state.

State property

See also

Determines a button's state—pressed or not pressed.

Version added

2000

Syntax

intRet = *object*.**State**

object.**State** = *intExpression*

intRet	**Integer**. The state of the button.
object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** object.
intExpression	Required **Integer**. The new state of the button.

Remarks

The **State** property can be one of the following constants declared by the Visio type library in **VisUIButtonState**.

Constant	Value	Description
visButtonUp	0	Button is not pressed
visButtonDown	-1	Button is pressed

StencilPaths property

Gets or sets the paths where Microsoft Visio looks for stencils.

Version added

4.0

Syntax

strRet = *object*.**StencilPaths**

object.**StencilPaths** = *pathsStr*

strRet	**String**. A text string containing a list of folders.
object	Required. An expression that returns an **Application** object.
pathsStr	Required **String**. A text string containing a list of folders. Use semicolons to separate individual folders in the path string.

Remarks

The string passed to and received from the **StencilPaths** property is the same string shown on the **File Paths** tab in the **Options** dialog box (click **Options** on

the **Tools** menu, and then click **File Paths**).

When the Visio application looks for stencils, it looks in all paths named in the **StencilPaths** property and all the subfolders of those paths. If you pass the **StencilPaths** property to the **EnumDirectories** method, it returns a complete list of fully qualified paths in which the Visio application looks.

If a path is not fully qualified, the Visio application looks for the folder in the folder that contains the Visio program files (*appObj*.**Path**). For example, if the Visio executable file is installed in c:\Visio, and the **StencilPaths** property is "Stencils;d:\Stencils", Visio looks for stencils in both c:\Visio\Stencils and d:\Stencils.

StatusBarItems property

See also Example Applies to

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property returned the **StatusBarItems** collection of a **StatusBar** or **StatusBarItem** object.

StatusBars property

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property returned the **StatusBars** collection of a **UIObject** object.

Style property

See also

Determines whether a toolbar button or menu item shows an icon, a caption, or some combination.

Version added

2000

Syntax

intRet = *object*.**Style**

object.**Style** = *intExpression*

intRet	**Integer**. The current style of a toolbar button or menu item.
object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** object.
intExpression	Required **Integer**. The new style for a toolbar button or menu item.

Remarks

If a style consists of different text, line, and fill styles, the **Style** property returns the fill style. If you set the **Style** property to a nonexistent style, your program generates an error.

Possible *intval* and *intExpression* values are listed in the following table. These constants are declared by the Visio type library in **VisUIButtonStyle**.

Constant	Value
visButtonAutomatic	0
visButtonCaption	1
visButtonIcon	2
visCaptionAndIcon	3

Style property (Cell object)

Example

Gets the style that contains a **Cell** object.

Version added

2.0

Syntax

objRet = *object*.**Style**

objRet	A **Style** object that represents the style containing the cell.
object	Required. An expression that returns a **Cell** object.

Remarks

If a style consists of different text, line, and fill styles, the **Style** property returns the fill style.

If a **Cell** object is in a style, its **Style** property returns the style that contains the cell, and its **Shape** property returns **Nothing**.

If a **Cell** object is in a shape, its **Shape** property returns the shape that contains the cell, and its **Style** property returns **Nothing**.

Style property (Section object)

Example

Gets the style that contains a **Section** object.

Version added

2.0

Syntax

objRet = *object*.**Style**

objRet	A **Style** object that represents the style containing the section.
object	Required. An expression that returns a **Section** object.

Remarks

If a style consists of different text, line, and fill styles, the **Style** property returns the fill style.

If a **Section** object is in a style, its **Style** property returns the style that contains the cell, and its **Shape** property returns **Nothing**.

If a **Section** object is in a shape, its **Shape** property returns the shape that contains the cell, and its **Style** property returns **Nothing**.

Style property (Shape object)

Example

Gets or sets the style for a **Shape** object.

Version added

2.0

Syntax

strRet = *object*.**Style**

object.**Style** = *stringExpression*

strRet	**String**. The fill style component of the style.
object	Required. An expression that returns a **Shape** object that has or gets the style.
stringExpression	Required **String**. The name of the style to apply.

Remarks

If a style consists of different text, line, and fill styles, the **Style** property returns the fill style. If you set the **Style** property to a nonexistent style, your program

generates an error.

To preserve local formatting, use the **StyleKeepFmt** property.

Beginning with Visio 2002, an empty string ("") will cause the master's style to be reapplied to the shape. (Earlier versions generate a "no such style" exception.) If the shape has no master, its style remains unchanged. Setting *stringExpression* to an empty string is the equivalent of selecting ***Use master's format*** in the **Text style**, **Line style**, or **Fill style** list in the **Style** dialog box (on the **Format** menu, click **Style**).

Style property (Selection object)

Example

Gets or sets the style for a **Selection** object.

Version added

2.0

Syntax

strRet = *object*.**Style**

object.**Style** = *stringExpression*

strRet	**String**. The fill style component of the style.
object	Required. An expression that returns a **Selection** object that has or gets the style.
stringExpression	Required **String**. The name of the style to apply.

Remarks

If a style consists of different text, line, and fill styles, the **Style** property returns the fill style. If you set the **Style** property to a nonexistent style, your program

generates an error.

To preserve local formatting, use the **StyleKeepFmt** property.

Beginning with Visio 2002, an empty string ("") will cause the master's style to be reapplied to the selection. (Earlier versions generate a "no such style" exception.) If the selection has no master, its style remains unchanged. Setting *stringExpression* to an empty string is the equivalent of selecting ***Use master's format*** in the **Text style**, **Line style**, or **Fill style** list in the **Style** dialog box (on the **Format** menu, click **Style**).

Style property (Row object)

Example

Gets the style that contains a **Row** object.

Version added

2.0

Syntax

objRet = *object*.**Style**

objRet	A **Style** object that represents the style containing the row.
object	Required. An expression that returns a **Row** object.

Remarks

If a style consists of different text, line, and fill styles, the **Style** property returns the fill style.

If a **Row** object is in a style, its **Style** property returns the style that contains the cell, and its **Shape** property returns **Nothing**.

If a **Row** object is in a shape, its **Shape** property returns the shape that contains the cell, and its **Style** property returns **Nothing**.

StyleKeepFmt property

See also Example

Applies a style to an object while preserving local formatting.

Version added

2.0

Syntax

object.**StyleKeepFmt** = *stringExpression*

object	Required. An expression that returns a **Shape** or **Selection** object that gets the style.
stringExpression	Required **String**. The name of the style to apply.

Remarks

Setting the **StyleKeepFmt** property is equivalent to selecting the **Preserve local formatting** check box in the **Style** dialog box (click **Style** on the **Format** menu). Setting a style to a nonexistent style generates an error.

Beginning with Visio 2002, an empty string ("") will cause the master's style to be reapplied to the selection or shape. (Earlier versions generate a "no such style" exception.) If the selection or shape has no master, its style remains unchanged. Setting *stringExpression* to an empty string is the equivalent of selecting *Use master's format* in the **Text style**, **Line style**, or **Fill style** list on the **Style** dialog box (on the **Format** menu, click **Style**).

Styles property

Returns the **Styles** collection for a document.

Version added

2.0

Syntax

objRet = *object*.**Styles**

objRet	The **Styles** collection of the **Document** object.
object	Required. An expression that returns a **Document** object.

SubAddress property

Gets or sets the subaddress in a shape's **Hyperlink** object.

Version added

5.0

Syntax

strRet = *object*.**SubAddress**

object.**SubAddress** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Hyperlink** object.
stringExpression	Required **String**. The new value for the field.

Remarks

Setting the **SubAddress** property of a shape's **Hyperlink** object is optional unless the **Address** property is blank. In this case the **SubAddress** must contain the name of the drawing page.

Setting a hyperlink's **Subaddress** property is equivalent to entering information in **Sub-address** box in the **Hyperlinks** dialog box (click **Hyperlinks** on the **Insert** menu). This is also equivalent to setting the result of the Subaddress cell in the shape's Hyperlink.Row row in the ShapeSheet window.

The **SubAddress** property for a **Hyperlink** object specifies a sublocation within the hyperlink's address. For Visio files, this can be a page name. For Microsoft Excel, this can be a worksheet or a range within a worksheet. For HTML pages, this can be a sub-anchor.

The hyperlink address for which a subaddress is being supplied must support subaddress linking.

Subject property

Gets or sets the value of the **Subject** field in a document's properties.

Version added

2.0

Syntax

strRet = *object*.**Subject**

object.**Subject** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The new value for the field.

Remarks

Setting the **Subject** property is equivalent to entering information in the **Subject** box in the **Properties** dialog box (click **Properties** on the **File** menu).

SubType property

Example

Returns the subtype of a **Window** object that represents a drawing window.

Version added

4.0

Syntax

intRet = object.**SubType**

intRet	**Integer**. The subtype of the **Window** object.
object	Required. An expression that returns a **Window** object.

Remarks

If the **Type** property of a **Window** object returns any value other than **visDrawing**, the **SubType** property returns the same value as the **Type** property. If the **Type** property of a **Window** object returns **visDrawing**, the **SubType** property returns one of the following values.

Constant	Value	Description

visPageWin	128	A drawing window showing a page.
visPageGroupWin	160	A group editing window of a group on a page.
visMasterWin	64	A master drawing page window.
visMasterGroupWin	96	A group editing window of a group in a master.

TableName property

See also Example

Gets or sets the name of an **AccelTable** object.

Version added

4.0

Syntax

strRet = *object*.**TableName**

object.**TableName** = *nameStr*

strRet	**String**. The current name of the object.
object	Required. An expression that returns an **AccelTable** object.
nameStr	Required **String**. The new name for the object.

Remarks

This property is not currently used.

TabPropsRow property

Example

Returns the index of the Tab Properties row that contains tab formatting information for a **Characters** object.

Version added

3.0

Syntax

intRet = *object*.**TabPropsRow**(*bias*)

intRet	**Integer**. The index of the row that defines the **Character** object's formatting.
object	Required. An expression that returns a **Characters** object.
bias	Required **Integer**. The direction of the search.

Remarks

You can retrieve rows that represent runs of tab formatting by specifying a row index as an argument to the **CellsSRC** property of a shape. You can also view or change tab formats on the **Tabs** tab (click **Text** on the **Format** menu, and then

click **Tabs**).

If the tab format for the **Characters** object is represented by more than one Tab Properties row, the **TabPropsRow** property returns -1. If the **Characters** object represents an insertion point rather than a sequence of characters (that is, if its **Begin** and **End** properties return the same value), use the *bias* argument to determine which row index to return.

Constant	Value
visBiasLetVisioChoose	0
visBiasLeft	1
visBiasRight	2

Specify **visBiasLeft** for the row that covers tab formatting for the character to the left of the insertion point. Use **visBiasRight** for the row that covers tab formatting for the character to the right of the insertion point.

Target property

Example

Gets or sets the target of an event.

Version added

4.0

Syntax

strRet = *object*.**Target**

object.**Target** = *stringExpression*

strRet	**String**. The current target.
object	Required. An expression that returns an **Event** object.
stringExpression	Required **String**. The target to set.

Remarks

An event consists of an event-action pair. When the event occurs, the action is performed. An event also specifies the target of the action and arguments to send to the target.

If the action code of the event is **visActCodeRunAddon**, the **Target** property contains the name of the add-on to run.

If the action code of the event is **visActCodeAdvise**, the **Target** property is not available. Attempting to get or set the **Target** property for such an event causes an exception.

TargetArgs property

Example

Gets or sets the arguments to be sent to the target of an event.

Version added

4.0

Syntax

strRet = *object*.**TargetArgs**

object.**TargetArgs** = *stringExpression*

strRet	**String**. The current arguments.
object	Required. An expression that returns an **Event** object.
stringExpression	Required **String**. The new arguments to set.

Remarks

An event consists of an event-action pair. When the event occurs, the action is performed. An event also specifies the target of the action and arguments to send to the target.

When you use **visActCodeRunAddon**, the **TargetArgs** property contains the arguments to send to the add-on when it is run.

When you use **visActCodeAdvise**, the **TargetArgs** property contains the string specified with the **AddAdvise** method when the **Event** object was created. When the program receives notification of the event, it can get the **Event** object and its **TargetArgs** property to obtain the string.

TargetDocumentName property

Example

Gets and sets the path and file name of a Microsoft Visio document (usually a stencil) that contains the master to which a master shortcut refers.

Version added

2000

Syntax

strRet = *object*.**TargetDocumentName**

object.**TargetDocumentName** = *strExpression*

strRet	**String**. Path and file name of the document.
object	Required. An expression that returns a **MasterShortcut** object.
strExpression	Required **String**. The new path and file name of the document.

Remarks

If the target document is moved, deleted, or renamed, or the property is set to the

path of a nonexistent file, the application will not be able to access the shortcut's target master. As a result, the end user will not be able to use the shortcut to drop shapes onto their drawing.

If the **TargetDocumentName** property contains a file name but no path, the application looks for the target document in the file path set on the **File Path** tab in the **Options** dialog box (click **Options** on the **Tools** menu, and then click **File Paths**). The name may refer either to the document's file name or to one of its alternate file names. To set an alternate name for a document, use the **AlternateNames** property.

The **TargetDocumentName** property does not support file names with relative paths.

TargetMasterName property

Example

Gets or sets the name of the master to which the master shortcut refers.

Version added

2000

Syntax

strRet = *object*.**TargetMasterName**

object.**TargetMasterName** = *strExpression*

strRet	**String**. The name of the master.
object	Required. An expression that returns a **MasterShortcut** object.
strExpression	Required **String**. The new name of the master.

Remarks

The name specified by this property must be the target master's universal name, not its localized name.

When the user drops a master shortcut onto a drawing page, the application first locates the document identified by the shortcut's **TargetDocumentName** property, then it searches that document for a master whose universal name matches the shortcut's **TargetMasterName** property. Once located, the target master (not the shortcut) is used to create the new shape instance on the drawing page.

Template property

See also

Returns the name of the template from which the document was created.

Version added

4.0

Syntax

strRet = *object*.**Template**

strRet	**String**. The name of the template from which the **Document** object was created.
object	Required. An expression that returns a **Document** object.

TemplatePaths property

Gets or sets the paths where Microsoft Visio looks for templates. To indicate more than one folder, separate individual items in the path string with semicolons.

Version added

4.0

Syntax

strRet = *object*.**TemplatePaths**

object.**TemplatePaths** = *pathsStr*

strRet	**String**. A text string containing a list of folders.
object	Required. An expression that returns an **Application** object.
pathsStr	Required **String**. A text string containing a list of folders.

Remarks

The string passed to and received from the **TemplatePaths** property is the same

string shown on the **File Paths** tab in the **Options** dialog box (click **Options** on the **Tools** menu, and then click **File Paths**).

When the application looks for templates, it looks in all paths named in the **TemplatePaths** property and all the subfolders of those paths. If you pass the **TemplatePaths** property to the **EnumDirectories** method, it returns a complete list of fully qualified paths in which the application looks.

If a path is not fully qualified, the application looks for the folder in the folder that contains the Visio program files (*appObj*.**Path**). For example, if the Visio executable file is installed in c:\Visio, and the **TemplatePaths** property is "Templates;d:\Templates", the Visio application looks for templates in both c:\Visio\Templates and d:\Templates.

Text property (Characters object)

Returns the range of text represented by a **Characters** object, which may be a subset of the shape's text depending on the values of the **Characters** object's **Begin** and **End** properties.

Version added

2.0

Syntax

strRet = *object*.**Text**

object.**Text** = *stringExpression*

strRet	**Variant**. The text of the **Characters** object returned in a **Variant** of type **String**.
object	Required. An expression that returns the **Characters** object that owns the text.
stringExpression	Required **Variant**. The text of the **Characters** object in a **Variant** of type **String**.

Remarks

The text for a **Characters** object is returned in a **Variant** of type **String**, as opposed to in a **String**. This is typically transparent if you're using Microsoft Visual Basic or Visual Basic for Applications. If you are using C/C++ and want a **String** rather than a **Variant**, use the **TextAsString** property.

In the text returned by a **Characters** object, fields are expanded to the number of characters that are visible in the drawing window. For example, if a shape's text contains a field that displays the file name of a drawing, the **Text** property of a **Characters** object returns the expanded file name (provided the **Begin** and **End** properties were not altered).

If a **Characters** object represents the text of a shape that is a group, it will always return the text of the group.

Objects from other applications and guides don't have a **Text** property.

Text property (Shape object)

Example

Returns all of the shape's text.

Version added

2.0

Syntax

strRet = *object*.**Text**

object.**Text** = *stringExpression*

strRet	**String**. The text of the **Shape** object returned as a string.
object	Required. An expression that returns a **Shape** object.
stringExpression	Required **String**. New text for the **Shape** object.

Remarks

In the text returned by the **Text** property of a **Shape** object, fields are represented by an escape character (30 (&H1E)) For example, if a **Shape** object's text contains a field that displays the file name of a drawing, the **Shape** object's **Text**

property returns an escape character where that field is inserted into the text. If you want the text to contain the expanded field, get the shape's **Characters** property, then get the **Text** property of the resulting **Characters** object.

If the shape is a group, the text returned is dependent on the value of the IsTextEditTarget cell.

If IsTextEditTarget is TRUE, then the **Text** property of the **Shape** object returns the text of the group.

If IsTextEditTarget is FALSE, then the **Text** property of the **Shape** object returns the text of the shape in the group at the top of the stacking order.

Objects from other applications and guides don't have a **Text** property.

TextAsString property

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property returned the range of text represented by a **Characters** object.

TextBasedOn property

Example

Gets or sets the text style on which a **Style** object is based.

Version added

4.0

Syntax

strVal = *object*.**TextBasedOn**

object.**TextBasedOn** = *styleName*

strVal	**String**. The name of the current based-on text style.
object	Required. An expression that returns a **Style** object.
styleName	Required **String**. The name of the new based-on style.

Remarks

To base a style on no style, set the **TextBasedOn** property to a zero-length string ("").

TextStyle property

Example

Gets or sets the text style for an object.

Version added

2.0

Syntax

strRet = *object*.**TextStyle**

object.**TextStyle** = *stringExpression*

strRet	**String**. The current text style.
object	Required. An expression that returns a **Shape** or **Selection** object.
stringExpression	Required **String**. The name of the text style to apply.

Remarks

Setting this property is equivalent to selecting a style from the **Text Style** list in Visio.

Setting a style to a nonexistent style generates an error. Setting one kind of style to an existing style of another kind (for example, setting the **TextStyle** property to a fill style) does nothing. Setting one kind of style to an existing style that has more than one set of attributes changes only the attributes for that component (for example, setting the **TextStyle** property to a style with line, text, and fill attributes changes only the text attributes).

To preserve a shape's local formatting, use the **TextStyleKeepFmt** property.

Beginning with Microsoft Visio 2002, an empty string ("") will cause the master's style to be reapplied to the selection or shape. (Earlier versions generate a "no such style" exception.) If the selection or shape has no master, its style remains unchanged. Setting *stringExpression* to an empty string is the equivalent of selecting **Use master's format** in the **Text style**, **Line style**, or **Fill style** list in the **Style** dialog box (on the **Format** menu, click **Style**).

TextStyleKeepFmt property

Example

Applies a text style to an object while preserving local formatting.

Version added

2.0

Syntax

object.**TextStyleKeepFmt** = *stringExpression*

object	Required. An expression that returns a **Shape** or **Selection** object.
stringExpression	Required **String**. The name of the style to apply.

Remarks

Setting the **TextStyleKeepFmt** property is equivalent to selecting the **Preserve local formatting** check box in the **Style** dialog box (click **Style** on the **Format** menu).

Setting a style to a nonexistent style generates an error. Setting one kind of style

to an existing style of another kind (for example, setting the **TextStyleKeepFmt** property to a fill style) does nothing. Setting one kind of style to an existing style that has more than one set of attributes changes only the attributes for that component (for example, setting the **TextStyleKeepFmt** property to a style with line, text, and fill attributes changes only the text attributes).

Beginning with Microsoft Visio 2002, an empty string ("") will cause the master's style to be reapplied to the selection or shape. (Earlier versions generate a "no such style" exception.) If the selection or shape has no master, its style remains unchanged. Setting *stringExpression* to an empty string is the equivalent of selecting **Use master's format** in the **Text style**, **Line style**, or **Fill style** list in the **Style** dialog box (on the **Format** menu, click **Style**).

Time property

Example

Returns the most recently recorded date and time.

Version added

2002

Syntax

dateRet = *object*.**Time**

dateRet	**Date**. The current date and time.
object	Required. An expression that returns a **Document** object.

Remarks

The **Time** property is updated whenever any values are updated in the following: the **TimeEdited** property, the **TimePrinted** property, the **TimeCreated** property, the **TimeSaved** property, or the NOW function.

The value to the left of the decimal point represents the date, and the value to the right of the decimal point represents the time.

TimeCreated property

Example

Returns the date and time the document was created.

Version added

2002

Syntax

dateRet = object.**TimeCreated**

dateRet	**Date**. The date and time the document was created.
object	Required. An expression that returns a **Document** object.

Remarks

The value to the left of the decimal point represents the date, and the value to the right of the decimal point represents the time.

TimeEdited property

Example

Returns the date and time the document was last edited.

Version added

2002

Syntax

dateRet = object.**TimeEdited**

dateRet	**Date**. The date and time the document was last edited.
object	Required. An expression that returns a **Document** object.

Remarks

The value to the left of the decimal point represents the date, and the value to the right of the decimal point represents the time.

TimePrinted property

Example

Returns the date and time the document was last printed.

Version added

2002

Syntax

dateRet = object.**TimePrinted**

dateRet	**Date**. The date and time the document was last printed.
object	Required. An expression that returns a **Document** object.

Remarks

The value to the left of the decimal point represents the date, and the value to the right of the decimal point represents the time.

TimeSaved property

Example

Returns the date and time the document was last saved.

Version added

2002

Syntax

dateRet = object.**TimeSaved**

dateRet	**Date**. The date and time the document was last saved.
object	Required. An expression that returns a **Document** object.

Remarks

The value to the left of the decimal point represents the date, and the value to the right of the decimal point represents the time.

Title property

Gets or sets the value of the **Title** field in a document's properties.

Version added

2.0

Syntax

strRet = *object*.**Title**

object.**Title** = *stringExpression*

strRet	**String**. The current value of the field.
object	Required. An expression that returns a **Document** object.
stringExpression	Required **String**. The new value for the field.

Remarks

Setting the **Title** property is equivalent to entering information in the **Title** box in the **Properties** dialog box (click **Properties** on the **File** menu).

ToCell property

Gets the cell to which a connection is made.

Version added

2.0

Syntax

objRet = object.**ToCell**

objRet	The **Cell** object to which the connection is made.
object	Required. An expression that returns a **Connect** object.

Remarks

A connection is defined by a reference in a cell in the shape from which the connection originates to a cell in the shape to which the connection is made. The **ToCell** property returns the **Cell** object to which the connection is made.

Following is a list of possible connections and their related **ToCell** property values.

- From the begin or end cell of a 1-D shape to…
- From the edge (a cell in the Alignment section) of a 2-D shape to…
- From an outward or inward/outward connection point cell of a 1-D shape to…
- From an outward or inward/outward connection point cell of a 2-D shape that is not a guide or guide point to…
- From a control handle to…

ToolbarItems property

Returns the **ToolbarItems** collection of a **Toolbar** object.

Version added

4.0

Syntax

objRet = object.**ToolbarItems**

objRet	The **ToolbarItems** collection of the **Toolbar** object.
object	Required. An expression that returns a **Toolbar** object.

Toolbars property

Returns the **Toolbars** collection of a **ToolbarSet** object.

Version added

4.0

Syntax

objRet = object.**Toolbars**

objRet	The **Toolbars** collection of the **ToolbarSet** object.
object	Required. An expression that returns a **ToolbarSet** object.

ToolbarSets property

Returns the **ToolbarSets** collection of a **UIObject** object.

Version added

4.0

Syntax

objRet = object.**ToolbarSets**

objRet	The **ToolbarSets** collection of the **UIObject** object.
object	Required. An expression that returns a **UIObject** object.

Remarks

If a **UIObject** object represents toolbars and status bars (for example, if the object was retrieved using the **BuiltInToolbars** property of an **Application** object), its **ToolbarSets** collection represents all of the toolbars for that **UIObject** object.

Use the **ItemAtID** property of a **ToolbarSets** object to retrieve toolbars for a

particular window context, for example, the drawing window. If a context does not include toolbars, it has no **ToolbarSets** collection.

ToolbarStyle property

See also Example Applies to

Beginning with Microsoft Visio 2002, this property is obsolete.

Remarks

In earlier versions, this property determined whether Microsoft Visio showed or hid a toolbar.

Top property

Gets the distance between the top of an object and the top of the docking area or the top of the screen if the object isn't docked; it sets the distance between the top of a **Menu** or **Toolbar** object and the top of the screen.

Version added

2000

Syntax

intRet = *object*.**Top**

object.**Top** = *intExpression*

intRet	**Integer**. The distance in pixels.
object	Required. An expression that returns a **MenuSet** or **Toolbar** object.
intExpression	Required **Integer**. The new distance in pixels.

ToPart property

Returns the part of a shape to which a connection is made.

Version added

2.0

Syntax

intRet = object.**ToPart**

intRet	**Integer**. The part of the shape to which a connection is made.
object	Required. An expression that returns a **Connect** object.

Remarks

The **ToPart** property identifies the part of a shape to which another shape is glued, such as its begin point or end point, one of its edges, or a connection point. The following constants declared by the Visio type library in member **VisToParts** show possible return values for the **ToPart** property.

Constant	Value

visConnectToError	-1
visToNone	0
visGuideX	1
visGuideY	2
visWholeShape	3
visGuideIntersect	4
visToAngle	7
visConnectionPoint	100 + row index of connection point

TopMargin property

Example

Specifies the top margin when printing a document.

Version added

4.0

Syntax

retVal = object.**TopMargin**([*units]*)

object.**TopMargin**([*units]*) = *newValue*

retVal	**Double**. The margin value expressed in the given units.
object	Required. An expression that returns a **Document** object.
units	Optional **Variant**. The units to use when retrieving or setting the margin value.
newValue	Required **Double**. The new margin value.

Remarks

If *units* is not provided, the **TopMargin** property will default to internal drawing

units (inches).

The **TopMargin** property corresponds to the **Top** setting in the **Print Setup** dialog box (on the **File** menu, click **Page Setup**, and then click **Setup** on the **Print Setup** tab).

Units can be an integer or string value such as "inches", "inch", "in.", or "i". Strings may be used for all supported Visio units such as centimeters, meters, miles, and so on. You can also use any of the units constants declared by the Visio type library in member **VisUnitCodes**.

For a list of valid integer and string values see About units of measure.

ToSheet property

Returns the shape to which one or more connections are made.

Version added

2.0

Syntax

objRet = *object*.**ToSheet**

objRet	The **Shape** object to which the connection is made.
object	Required. An expression that returns a **Connect** object or **Connects** collection.

Remarks

The **ToSheet** property for a **Connect** object always returns the shape to which the connection is made.

The **Connects** collection represents several connections. If every connection represented by the collection is made to the same shape, the **ToSheet** property

returns that shape. Otherwise, it returns **Nothing** and does not raise an exception.

TraceFlags property

See also

Gets or sets events logged during a Microsoft Visio instance.

Version added

5.0

Syntax

intRet = *object*.**TraceFlags**

object.**TraceFlags** = *intExpression*

intRet	**Long**. Current trace flags.
object	Required. An expression that returns an **Application** object.
intExpression	Required **Long**. New trace flags.

Remarks

The value of the **TraceFlags** property can be a combination of the following values.

Constant	Value	Description
visTraceEvents	&H1	Event occurrences
visTraceAdvises	&H2	Outgoing advise calls
visTraceAddonInvokes	&H4	Add-on invocations
visTraceCallsToVBA	&H8	VBA invocations

The **visTraceEvents** flag causes the Immediate window to log most Visio events as they happen. In most cases this occurs even if no external agent is listening or responding to the event. In a few cases, Visio knows there is no listener for an event and does not log those events. Visio also does not log idle events or advises. In addition, some events are specializations of other events and aren't recorded. For example, the **SelectionAdded** event is manufactured from distinct **ShapeAdded** events, so the Immediate window records the **ShapeAdded** events but not the **SelectionAdded** events.

Here is a string Visio might log when **visTraceEvents** is selected:

```
-event: 0x8040 /doc=1 /page=1 /shape=Sheet.1
```

The number after -event: is the code of the event that occurred. In this case 0x8040 is the code for the **ShapeAdded** event. The text following the event code differs from event to event.

The **visTraceAdvises** flag writes a line to the Immediate window just before Visio calls an event handler procedure, and another line just after the event handler returns. This includes event procedures in VBA projects, for example, procedures in **ThisDocument**. Here is an example of what you might see:

```
>advise seq=4 event=0x8040 sink=0x40097598
>advise seq=4
```

These strings indicate the call to and return from an event handler. The sequence number also indicates this event was the fourth one fired by Visio. The code of the event is 0x8040 and the address of the interface Visio called is 0x40097598.

The **visTraceAddonInvokes** flag records when Visio invokes an EXE or VSL add-on, and when Visio regains control. Here is an example:

>invokeAO: SHOWARGS.EXE
<invokeAO: completed

The **visTraceAddonInvokes** flag also traces attempts to invoke add-ons that are not present. For example, if a cell's formula is =RunAddon("xxx") and there is no add-on named "xxx", then the message "InvokeAO: Failed to map 'xxx' to known Add-on" is logged.

The **visTraceCallToVBA** flag writes a line to the Immediate window just before it makes a call to VBA other than a call to an event procedure (use **visTraceAdvises** to log calls to VBA event procedures), and another line just after VBA returns control to Visio. This flag traces macro invocations, calls to VBA procedures resulting from evaluation of cells that make use of RunAddon or CallThis operands, and calls resulting from selection of custom menu or toolbar items. Here is an example:

>invokeVBA: Module1.MyMacro
<invokeVBA: completed

A message doesn't appear in the Immediate window unless a document with a VBA project is open. Visio queues a small number of messages to log when such a document opens. However, messages are lost if no document with a project is available for lengthy periods. Messages are also lost if VBA resets or if there are undismissed breakpoints.

Code in VBA projects can intersperse their messages with those logged by Visio using standard Debug.Print statements. Code in non-VBA projects can log messages to the VBA Immediate window using Document.VBProject.ExecuteLine("Debug.Print ""somestring""").

The **TraceFlags** property is recorded in the **TraceFlags** entry of the **Application** section of the registry.

Type property

Returns the type of the object.

Version added

2.0

Syntax

retVal = *object*.**Type**

retVal	**Integer**. The type of the **Shape** or **Window** object.
object	Required. An expression that returns a **Shape** or **Window** object.

Remarks

- Type values for Shape objects
- Type values for Window objects

TypelibMajorVersion property

Example

Returns the major version number of the Microsoft Visio type library.

Version added

2000

Syntax

intRet = object.**TypelibMajorVersion**

intRet	**Integer**. The major version number of the Visio type library.
object	Required. An expression that returns an **Application** object.

Remarks

The major and/or minor version number of the Visio type library will increase whenever the Visio type library is extended. A program can use the **TypelibMajorVersion** and **TypelibMinorVersion** properties to guarantee that the Visio version it is working with provides support for the features it is using.

Small changes to the Visio type library do not affect the **Application** object's

Version property.

TypelibMinorVersion property

Example

Returns the minor version number of the Microsoft Visio type library.

Version added

2000

Syntax

intRet = *object*.**TypelibMinorVersion**

intRet	**Integer**. The minor version number of the Visio type library.
object	Required. An expression that returns an **Application** object.

Remarks

The major and/or minor version number of the Visio type library will increase whenever the Visio type library is extended. A program can use the **TypelibMajorVersion** and **TypelibMinorVersion** properties to guarantee that the Visio version it is working with provides support for the features it is using.

Small changes to the Visio type library do not affect the **Application** object's **Version** property.

TypeSpecific1 property

Example

Gets or sets the type of a menu or toolbar item.

Version added

4.0

Syntax

intVal = *object*.**TypeSpecific1**

object.**TypeSpecific1** = *intExpression*

object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** object.
intVal, *intExpression*	Required **Integer**. The type of the menu, menu item, or toolbar item.

Remarks

The value of an object's **TypeSpecific1** property depends on the value of its **CntrlType** property.

CntrlType value	TypeSpecific1 value
visCtrlTypeBUTTON	Any constant prefixed with **visIconIX** that is declared by the Visio type library
visCtrlTypeEDITBOX **visCtrlTypeCOMBOBOX**	Zero (0)
visCtrlTypeLABEL	Any constant prefixed with **visStrID** that is declared by the Visio type library

TypeSpecific2 property

Example

Gets or sets the type of a menu, menu item, or toolbar item.

Version added

4.0

Syntax

intVal = object.**TypeSpecific2**

object.**TypeSpecific2** = *intExpression*

intVal	**Integer**. The type of menu, menu item, or toolbar.
object	Required. An expression that returns a **Menu**, **MenuItem**, or **ToolbarItem** object.
intExpression	Required **Integer**. The new type of menu, menu item, or toolbar item.

Remarks

The value of an object's **TypeSpecific2** property depends on the value of its **CntrlType** property.

CntrlType value	TypeSpecific1 value
visCtrlTypeBUTTON	The **TypeSpecific2** property is not used.
visCtrlTypeEDITBOX **visCtrlTypeCOMBOBOX**	The current width of the control expressed in pixels.
visCtrlTypeLABEL	The **TypeSpecific2** property is not used.

UndoEnabled property

Determines whether undo information is maintained in memory.

Version added

2000

Syntax

boolVal = *object*.**UndoEnabled**

object.**UndoEnabled** = *boolExpression*

boolVal	**Boolean**. **True** if the property is enabled; **False** if it is not.
object	Required. An expression that returns an **Application** object.
boolExpression	Required **Boolean**. **True** to enable undo; otherwise, **False**.

Remarks

When Visio starts, the value of the **UndoEnabled** property is **True**. Setting the value of the **UndoEnabled** property to **False** discontinues the collection of undo information in memory and purges the existing undo information.

An attempt should be made to maintain the property at its current value across the complete operation that you perform. In other words, use code structured like this:

```
bPrev = Application.UndoEnabled
Application.UndoEnabled = False
'large operation here
Application.UndoEnabled = bPrev
```

UniqueID property (Master object)

Example

Returns the unique ID of a master.

Version added

4.0

Syntax

strRet = object.**UniqueID**

strRet	**String**. The unique ID of the **Master** object.
object	Required. An expression that returns a **Master** object.

Remarks

A **Master** object always has a unique ID. If you copy a master, the new master has the same unique ID as the original master (as well as the same base ID). However, if you change the new master, a new unique ID is assigned but the base ID remains the same.

For details about the base ID, see the **BaseID** property.

You can determine a **Master** object's unique ID using the following:

```
idStr = mastObj.UniqueID
```

The value it returns is a string in the following form:

```
{2287DC42-B167-11CE-88E9-0020AFDDD917}
```

You can access a master by its unique ID using *Masters*.**Item**(*uniqueIDString*).

UniqueID property (Shape object)

Example

Returns or clears the unique ID of a shape.

Version added

4.0

Syntax

strRet = *object*.**UniqueID**(*flag*)

strRet	**String**. The unique ID of the **Shape** object.
object	Required. An expression that returns a **Shape** object.
flag	Required **Integer**. Gets, assigns, or clears the unique ID of a **Shape** object.

Remarks

By default, a shape does not have a unique ID. A shape acquires a unique ID only if you set its **UniqueID** property.

If a **Shape** object has a unique ID, no other shape in any other document will

have the same ID.

The *flag* argument controls the behavior of the **UniqueID** property. It should have one of the following values.

Constant	Value
visGetGUID	0
visGetOrMakeGUID	1
visDeleteGUID	2

The constant **visGetGUID** returns the unique ID string only if the shape already has a unique ID. Otherwise it returns a zero-length string ("").

The constant **visGetOrMakeGUID** returns the unique ID string of the shape. If the shape does not yet have a unique ID, it assigns one to the shape and returns the new ID.

The constant **visDeleteGUID** clears the unique ID of a shape and returns a zero-length string ("").

You can access a shape with its unique ID using *Shapes*.**Item**(*uniqueIDString*).

Units property

See also Example

Indicates the unit of measure associated with a **Cell** object.

Version added

3.0

Syntax

intRet = *object*.**Units**

intRet	**Integer**. The units associated with a cell's current value.
object	Required. An expression that returns a **Cell** object.

Remarks

The **Units** property can be used to determine the unit of measure currently associated with a cell's value. The various unit codes are declared by the Visio type library in member **VisUnitCodes**. For example, a cell's width might be expressed in inches (**visInches**) or in centimeters (**visCentimeters**). In some cases a program might behave differently depending on whether a cell's value is in metric or in imperial units.

For a list of valid unit codes, see About units of measure.

UserName property

See also Example

Gets or sets the user name of an **Application** object.

Version added

4.0

Syntax

strRet = *object*.**UserName**

object.**UserName** = *strExpression*

strRet	**String**. The user name.
object	Required. An expression that returns an **Application** object.
strExpression	Required **String**. The user name.

Remarks

Setting the **UserName** property is equivalent to entering a name in the **User name** box on the **General** tab in the **Options** dialog box (click **Options** on the **Tools** menu, and then click **General**).

VBE property

object;DAR_Objects_(A-M)_1015.htm">

Gets the root object of the object model exposed by Microsoft Visual Basic for Applications (VBA). Use this property to access and manipulate the VBA projects associated with currently open Microsoft Visio documents.

Version added

4.5

Syntax

objRet = *object*.**VBE**

objRet	Programmable object that exposes VBA methods and properties.
object	Required. An expression that returns an **Application** object.

Remarks

To get information about the object returned by the **VBE** property:

On the **Tools** menu, point to **Macros**, and then click **Visual Basic Editor**.

In the Visual Basic Editor, on the **Tools** menu, click **References**.

In the **References** dialog box, click **Microsoft Visual Basic for Applications Extensibility**, and then click **OK**.

On the **View** menu, click **Object Browser**.

In the **Project/Library** list, select the **VBIDE** type library.

In the **Classes** list, examine the class named **VBE**.

Beginning with Microsoft Visio 2002, the **VBE** property raises an exception if you are running in a secure environment and your system administrator has blocked access to the VBA object model. There is no user interface or programmatic way to turn this on—the system administrator must turn on (or off) access via a Group Policy. This is a protection mechanism against viruses that spread by accessing the Visual Basic projects in commonly used templates and injecting the virus code into them.

VBProject property

Returns a programmable object through which the Microsoft Visual Basic for Applications (VBA) project of the document can be controlled.

Version added

4.5

Syntax

objRet = *object*.**VBProject**

objRet	Programmable object that exposes methods and properties of the document's VBA project.
object	Required. An expression that returns a **Document** object.

Remarks

To get information about the object returned by the **VBProject** property:

On the **Tools** menu, point to **Macros**, and then click **Visual Basic Editor**.

In the Visual Basic Editor, on the **Tools** menu, click **References**.

In the **References** dialog box, click **Microsoft Visual Basic for Applications Extensibility**, and then click **OK**.

On the **View** menu, click **Object Browser**.

In the **Project/Library** list, select the **VBIDE** type library.

In the **Classes** list, examine the class named **VBProject**.

Beginning with Microsoft Visio 2002, the **VBProject** property raises an exception if you are running in a secure environment and your system administrator has blocked access to the Visual Basic object model. There is no user interface or programmatic way to turn this on—the system administrator must turn on (or off) access via a Group Policy. This is a protection mechanism against viruses that spread by accessing the Visual Basic projects in commonly used templates and injecting the virus code into them.

VBProjectData property

See also

Returns the Microsoft Visual Basic project data stored with a document.

Version added

2002

Syntax

valRet = object.**VBProjectData**

valRet	**Byte**. An array of data stored with a document's Visual Basic project.
object	Required. An expression that returns a **Document** object.

Example

You can use the **VBProjectData** property to determine whether a document has a project. The following macro demonstrates getting a reference to a document in Visio to determine whether the document has a project. The code runs from a program outside of the Visio document.

```
Private Sub Form_Load()
    'Declare document variable
    'and Array variable to hold project data
    Dim docObj As Object
    Dim projdata() As Byte
    'Get the first object in the Documents collection
    'of this instance of Visio
    Set docObj = GetObject(, "Visio.Application").Docume
    'Populate the array with project data
    projdata = docObj.VBProjectData
    Debug.Print LBound(projdata); UBound(projdata)
End Sub
```

If the document had no project associated with it, "0 –1" would be reported in the Immediate window. If the document had a project, the upper bound would be some number greater than zero (0). For example, "0 1535" would indicate that a project had 1,536 bytes of data.

Version property (Application object)

Returns the version of a running Microsoft Visio instance.

Version added

2.0

Syntax

strRet = *object*.**Version**

strRet	**String**. The Visio major and minor version numbers.
object	Required. An expression that returns an **Application** object.

Remarks

Use the **Version** property of the **Application** object to verify the version of a particular Visio instance. This information is helpful if your program requires a particular version. Both the major and minor version numbers are returned. The string returned by Microsoft Visio 2002 is "10.0".

Version property (Document object)

See also Example

Determines the version of a saved document.

Version added

2.0

Syntax

intRet = *object*.**Version**

object.**Version** = *intExpression*

intRet	**VisDocVersions**. The file format version the document is saved in.
object	Required. An expression that returns a **Document** object.
intExpression	Required **VisDocVersions**. The file format version in which to save the document.

Remarks

Setting the **Version** property of a document tells Visio which file format version to save the document in the next time the document is saved. The Visio type library declares constants for file format versions in **VisDocVersions**.

Visio 2002 can save the following versions.

Constant	Value	Description
visVersion50	&H50000	Visio 5.0 document

visVersion60	&H60000	Visio version 2000 or 2002 document

When Visio opens a document that was saved in an earlier version format, it converts the document's in-memory representation to the current version. However, when closing the document, Visio recognizes that the document was saved in an earlier version format and allows the user to choose the version in which to save the document.

ViewFit property

Example

Determines which auto-fit mode a window is in, if any.

Version added

2000

Syntax

longRet = *object*.**ViewFit**

object.**ViewFit** = *longVal*

longRet	**Long**. A constant that identifies the window mode.
object	Required. An expression that returns a **Window** object.
longVal	**Long**. A constant that identifies the new window mode.

Remarks

The **ViewFit** property applies to drawing windows only, and can have the following values.

Constant	Value
visFitNone	0
visFitPage	1
visFitWidth	2

If the value of the window's **Type** property is not **visDrawing**, then the **ViewFit** property returns **visFitNone**. Attempting to set the **ViewFit** property of this type of window raises an exception.

Visible property

Determines whether an object is visible.

Version added

2000

Syntax

boolVal = object.**Visible**

object.**Visible** = *boolExpression*

boolVal	**Boolean**. **True** if the object is visible; **False** if the object is not visible.
object	Required. An expression that returns an object in the **Applies to** list.
boolExpression	Required **Boolean**. **True** if the object is visible; **False** if the window is not visible.

Width property

Example

Gets the width of an object.

Version added

2000

Syntax

intLong = *object*.**Width**

intLong	**Long**. The width in pixels.
object	Required. An expression that returns an object in the **Applies to** list.

WindowHandle32 property

Returns the 32-bit handle of a Microsoft Visio window.

Version added

4.0

Syntax

retVal = object.**WindowHandle32**

retVal	**Long**. The **HWND** of the object's window.
object	Required. An expression that returns an **Application** or **Window** object.

Remarks

The **WindowHandle32** property of an **Application** object returns one of the following:

The **HWND** for the main Visio (frame) window (most common).

The **HWND** for the container application's main frame window if Visio is running in-place and active.

The **HWND** for the window returned by the **GetActiveWindow**() function if either frame window is disabled (for example, if a modal dialog box is running), For details about the **GetActiveWindow** function, see the Microsoft Platform SDK on the Microsoft Developer Network (MSDN) Web site.

Use the **WindowHandle32** property of the **Window** object to obtain the **HWND** for a window in the **Windows** collection of a Visio instance.

You can use the obtained **HWND** in Windows API calls.

Note Calls to the **WindowHandle** property (now hidden) are directed to the **WindowHandle32** property.

Windows property

object;DAR_Objects_(A-M)_1015.htm">

Returns the **Windows** collection for a Microsoft Visio instance or window.

Version added

2.0

Syntax

objRet = object.**Windows**

objRet	The **Windows** collection of the **Application** object.
object	Required. An expression that returns an **Application** or **Window** object that owns the collection.

WindowState property

Gets or sets the state of a window.

Version added

2000

Syntax

intRet = *object*.**WindowState**

object.**WindowState** = *intExpression*

intRet	**Long**. A constant that identifies the state of the window.
object	Required. An expression that returns a **Window** object.
intExpression	Required **Long**. The new state of the window.

Remarks

The values of *intRet* and *intExpression* can be a combination of the following constants, which are declared in the Visio type library in **VisWindowStates**.

Note The *varFlags* parameter to the **Add** method for the **Windows** collection can be composed of the various bits of **VisWindowStates**.

- VisWindowStates constants

wParam property

Example

Gets or sets the **wParam** field of the **MSG** structure being wrapped.

Version added

2002

Syntax

intRet = *object*.**wParam**

object.**wParam** = *intExpression*

intRet	**Long**. Additional information about the message that is dependent on the message number.
object	Required. An expression that returns a **MSGWrap** object.
intExpression	Required **Long**. The new value of the **wParam** field.

Remarks

The **wParam** property corresponds to the **wParam** field in the **MSG** structure defined as part of the Microsoft Windows operating system. If an event handler

is handling the **OnKeystrokeMessageForAddon** event, Visio passes a **MSGWrap** object as an argument when this event fires. A **MSGWrap** object is a wrapper around the Windows **MSG** structure.

For details, search for "MSG structure" on the Microsoft Developer Network (MSDN) Web site.

Zoom property

See also Example

Gets or sets the current display size (magnification factor) for a page in a window.

Version added

2.0

Syntax

retVal = *object*.**Zoom**

object.**Zoom** = *newZoom*

retVal	**Double**. The current display size for the window.
object	Required. An expression that returns a **Window** object.
newZoom	Required **Double**. The new display size for the window.

Remarks

Valid values range from 0.05 to 9.99 (5% to 999%). The value -1 fits the page into the window.

New cells (alphabetic list)

Cells that have been added to the Microsoft Visio 2002 ShapeSheet are listed in the following table (sorted alphabetically).

New cell	Section
ConLineRouteExt	Shape Layout
FillBkgndTrans	Fill Format
FillForegndTrans	Fill Format
LineColorTrans	Line Format
LineRouteExt	Page Layout
PlaceFlip	Page Layout
ShapePlaceFlip	Shape Layout
ShdwBkgndTrans	Fill Format
ShdwForegndTrans	Fill Format
TextBkgndTrans	Text Block Format
Transparency	Characters
Transparency	Layers

New cells (by section)

Cells that have been added to the Microsoft Visio 2002 ShapeSheet are listed in the following table (sorted by section).

Section	New cell
Characters	Transparency
Fill Format	FillBkgndTrans, FillForegndTrans, ShdwBkgndTrans, ShdwForegndTrans
Layers	Transparency
Line Format	LineColorTrans
Page Layout	LineRouteExt, PlaceFlip
Shape Layout	ConLineRouteExt, ShapePlaceFlip
Text Block Format	TextBkgndTrans

New objects

The following new object has been added to Microsoft Visio 2002.

New object	Description
MSGWrap	Used with the OnKeystrokeMessageForAddon event to provide access to Windows keystroke messages.

New properties (alphabetic list)

Properties that have been added to Microsoft Visio 2002 are listed in the following table (sorted alphabetically).

New property	Object
ActivePrinter	**Application**
AllowEditing	**Window**
AvailablePrinters	**Application**
BeginGroup	**MenuItem**, **ToolbarItem**
Build	**Application**
BuildNumberCreated	**Document**
BuildNumberEdited	**Document**
COMAddins	**Application**
CommandBars	**Application**
ContainsWorkspace	**Document**
DefaultAngleUnits	**Application**
DefaultDurationUnits	**Application**
DefaultGuideStyle	**Document**
DefaultPageUnits	**Application**
DefaultTextUnits	**Application**
DynamicGridEnabled	**Document**
EditCopy	**Master**
EmailRoutingData	**Document**
FooterCenter	**Document**
FooterLeft	**Document**

FooterMargin	**Document**
FooterRight	**Document**
ForeignData	**Shape**
FullBuild	**Application**
FullBuildNumberCreated	**Document**
FullBuildNumberEdited	**Document**
GlueEnabled	**Document**
GlueSettings	**Document**
HeaderCenter	**Document**
HeaderFooterColor	**Document**
HeaderFooterFont	**Document**
HeaderLeft	**Document**
HeaderMargin	**Document**
HeaderRight	**Document**
hwnd	**MSGWrap**
Icon	**Master**, **MasterShortcut**, **Window**
InheritedFormulaSource	**Cell**
InheritedValueSource	**Cell**
InhibitSelectChange	**Application**
InPlace	**Document**, **Window**
IsChanged	**Master**
lParam	**MSGWrap**
MasterShape	**Shape**
MergeCaption	**Window**
MergeClass	**Window**
MergeID	**Window**
MergePosition	**Window**
message	**MSGWrap**
Original	**Master**
PageTabWidth	**Window**
Password	**Document**
Picture	**Master**, **Page**, **Selection**, **Shape**
posttime	**MSGWrap**
PreviewPicture	**Document**

PrintCopies	**Document**
Printer	**Document**
PrintTileCount	**Page**
ptx	**MSGWrap**
pty	**MSGWrap**
RootShape	**Shape**
ShowPageTabs	**Window**
ShowScrollBars	**Window**
SnapAngles	**Document**
SnapEnabled	**Document**
SnapExtensions	**Document**
SnapSettings	**Document**
SolutionXMLElement	**Document**
SolutionXMLElementCount	**Document**
SolutionXMLElementExists	**Document**
SolutionXMLElementName	**Document**
Time	**Document**
TimeCreated	**Document**
TimeEdited	**Document**
TimePrinted	**Document**
TimeSaved	**Document**
VBProjectData	**Document**
wParam	**MSGWrap**

New properties (by object)

Properties that have been added to Microsoft Visio 2002 are listed in the following table (sorted by object).

Object	New property
Application	ActivePrinter, Build, COMAddins, CommandBars, DefaultAngleUnits, DefaultDurationUnits, DefaultGuideStyle, DefaultPageUnits, DefaultTextUnits, DynamicGridEnabled, FullBuild, InhibitSelectChange
Cell	InheritedFormulaSource, InheritedValueSource
Document	AvailablePrinters, BuildNumberCreated, BuildNumberEdited, ContainsWorkspace, FooterCenter, FooterLeft, FooterMargin, FooterRight, FullBuildNumberCreated, FullBuildNumberEdited, GlueEnabled, GlueSettings, HeaderCenter, HeaderFooterColor, HeaderFooterFont, HeaderLeft, HeaderMargin, HeaderRight, InPlace, Password, PreviewPicture, PrintCopies, Printer, SnapAngles, SnapEnabled, SnapExtensions, SnapSettings, SolutionXMLElement,

	SolutionXMLElementCount, SolutionXMLElementExists, SolutionXMLElementName, Time, TimeCreated, TimeEdited, TimePrinted, TimeSaved, VBProjectData
Master	EditCopy, Icon, IsChanged, Original, Picture
MasterShortcut	Icon
MenuItem	BeginGroup
MSGWrap	hwnd, lParam, message, posttime, ptx, pty, wParam
Page	Picture, PrintTileCount
Selection	Picture
Shape	ForeignData, MasterShape, Picture, RootShape
ToolbarItem	BeginGroup
Window	AllowEditing, AvailablePrinters, Icon, InPlace, MergeCaption, MergeClass, MergeID, MergePosition, PageTabWidth, ShowPageTabs, ShowScrollBars

New methods (alphabetic list)

Methods that have been added to Microsoft Visio 2002 are listed in the following table (sorted alphabetically).

New method	Object
CopyPreviewPicture	**Document**
DeleteSolutionXMLElement	**Document**
GetFilterCommands	**Event**
GetFilterObjects	**Event**
GetFilterSRC	**Event**
PasteSpecial	**Master**, **Page**, **Shape**
PrintTile	**Page**
RenameCurrentScope	**Application**
ResizeToFitContents	**Master**, **Page**
SetFilterCommands	**Event**
SetFilterObjects	**Event**
SetFilterSRC	**Event**
SwapEnds	**Selection**, **Shape**

New methods (by object)

Methods that have been added to Microsoft Visio 2002 are listed in the following table (sorted by object).

Object	New method
Application	RenameCurrentScope
Document	CopyPreviewPicture, DeleteSolutionXMLElement
Event	GetFilterCommands, GetFilterObjects, GetFilterSRC, SetFilterCommands, SetFilterObjects, SetFilterSRC
Master	PasteSpecial, ResizeToFitContents
Page	PasteSpecial, PrintTile, ResizeToFitContents
Selection	SwapEnds
Shape	PasteSpecial, SwapEnds

New events

The following new event has been added to Microsoft Visio 2002.

New event	Object
OnKeystrokeMessageForAddon	**Application**, **Window**

PrintCopies property

Example

Specifies the number of copies to print.

Version added

2002

Syntax

intRet = *object*.**PrintCopies**

object.**PrintCopies** = *intValue*

intRet	**Long**. The current number of copies.
object	Required. An expression that returns a **Document** object.
intValue	Required **Long**. The new number of copies.

Remarks

The **PrintCopies** property corresponds to the **Number of copies** box in the **Print** dialog box (click **Print** on the **File** menu).